Technical Report 1284

A Parallel Crossbar Routing Chip for a Shared Memory Multiprocessor

Henry Minsky

MIT Artificial Intelligence Laboratory

This blank page was inserted to preserve pagination.

A Parallel Crossbar Routing Chip for a Shared Memory Multiprocessor

by

Henry Minsky

B.S.,Massachusetts Institute of Technology (1984)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 1991

Abstract

This thesis describes the design and implementation of an integrated circuit and associated packaging to be used as the building block for the data routing network of a large scale shared memory multiprocessor system.

A general purpose multiprocessor depends on high-bandwidth, low-latency communications between computing elements. This thesis describes the design and construction of RN1, a novel self-routing, enhanced crossbar switch as a CMOS VLSI chip. This chip provides the basic building block for a scalable pipelined routing network with byte-wide data channels. A series of RN1 chips can be cascaded with no additional internal network components to form a multistage fault-tolerant routing switch. The chip is designed to operate at clock frequencies up to 100Mhz using Hewlett-Packard's HP34 1.2μ process. This aggressive performance goal demands that special attention be paid to optimization of the logic architecture and circuit design.

Thesis Supervisor: Thomas Knight, Jr.
Title: Asst. Professor, Dept. Electrical Engineering and Computer Science

This empty page was substituted for a blank page in the original document.

Contents

List of Figures

List of Tables

Acknowledgements

I would first of all like to thank Tom Knight, my thesis advisor, without whose support and encouragement I would not have been able to get this far. His vision of engineering in general, and large computer systems in particular, created the framework in which this work was done. The deceptive simplicity of the router protocol is a trademark of Prof. Knight's economical design style. I would next like to thank André DeHon, who has been instrumental in the evolution of this design. André collaborated in the crucial early phase of the design, and contributed many important design ideas and improvements. His encouragement was especially valuable during the long hours of optimization, redesign, and verification which this design entailed. Numerous discussions and arguments have greatly contributed to my understanding of computer design. I also thank him for encouraging me to get this thesis done, finally. Fred Drenckhahn contributed his talent and design wizardry (and good cheer) to actually produce all of the crucial packaging and connector technology.

I want to thank the AI Laboratory for existing in its present form, which means thanking many of the cool laboratory directors and professors; Winston, Tomas Lozano Perez, Eric Grimson, Rod Brooks, Marc Raibert, and of course my father Marvin Minsky, for creating the lab and me. The AI Lab is still a place where you can follow your imagination to wherever it leads you. The many 7th floor denizens, including Ian Horswill, Paul Viola, Alan Bawden, Sandy Wells all helped out at various times with good conversations or crucial insights. Most of all, thanks to my wife Milan, who was always there when I needed her.

This research was done at and supported by the Massachusetts Institute of Technology Artificial Intelligence Laboratory. Support for the Artificial Intelligence Laboratory is provided in part by the Advanced Research Projects Agency under the Office of Naval Research contracts ONR N00014-88-K-0825 and N00014-85-K-0124.

Chapter 1

Introduction

This thesis describes the design of **RN1**, a VLSI chip and associated packaging used to construct a multistage interprocessor communication switch called the Transit network. The Transit network is used to provide a data communication substrate on which to build a shared-memory multiprocessor.

This chapter discusses the general goals of the Transit project, and the need for an interprocessor switching network. Chapter 2 provides an overview of the range of switching network design choices. Chapter 3 introduces the RN1 crossbar chip. Chapter 4 discusses the RN1 chip's communication protocols in depth. Chapter 5 details the internal architecture of the chip. Chapter 6 goes into more detail about high-performance VLSI circuit approaches. Finally, Chapter 7 provides an analysis of chip performance, bugs, and future improvements.

1.1 Building a Multi-model Parallel Processor

The goal of the Transit project is to build a family of moderate to large scale general purpose multiprocessor systems. These systems will initially have from 64 to 256 MIMD processors, currently commerical RISC microcontrollers, communicating with each other in a configuration which will appear to the processors as a global shared memory space. In such a machine, interprocessor communication can be viewed as a special case of memory access (or vice-versa).

The architecture of the Transit machine is designed to support multiple models of parallel computation. At its core is a high-bandwidth low-latency interprocessor communication net-

work. This network will allow efficient implementations of a coherent shared-memory model of computation, a message passing model, a dataflow model, systolic arrays, and even an efficient SIMD model. Each of these computational models depends on a different mixture of bandwidth and latency communication between processing elements.

Existing parallel architectures tend to force the users into a specific programming model. The Transit machine is designed to present the programmer with a fast, simple hardware platform with the primitive operations on which to build a parallel computation application. The trend toward RISC processor architectures is instructive. RISC architectures moved many of the monolithic uniprocessor computational primitives (function call, complex addressing-mode memory references, exception handling) from hardware to software, allowing the compiler writers and programmers to utilize the hardware in a more efficient fashion. The higher performance of the RISC systems over CISC is due in part to the decreased cycle time gained by simpler control paths. But gains in software performance of RISC systems are also due to the flexibility gained by freeing the programmer or compiler writer from being locked into a specific high-level hardware-enforced method of serial computation. Similarly, the Transit architecture is designed to allow writers of parallel processing software to program a machine which supports a set of fast, high-bandwidth data communication primitives, without being committed to a single higher-level computational model.

1.2 The Need for a Switching Network

For any system with reasonably heavy memory access patterns and processors numbering more than about eight, the data bandwidth of a single bus, no matter how wide, becomes insufficient to provide the access needed by all processors. If there are n processors which all wish to use the bus, the bus can, on average, provide each one with access only $1/n$ of the time. For computations with frequent memory access and large communication bandwidth, more data channels are needed to connect the processors.

A ring topology with n hops can, in the best case, increase the available bus bandwidth by a factor of n over a simple bus, but the latency is also increased by n. Also, the increased bandwidth is not worth as much as it seems because messages which circulate for more than one hop on the ring tie up bus resources as they travel.

A higher performance and more general approach to processor interconnection method is the switching network. This is a core fabric for communication. A switching network can be thought of as a box, with a set of input and output ports, which provides the service of passing data from any input to any output. Chapter 2 provides an overview of switching network concepts.

1.3 Interprocessor Switching Network Design Goals

To provide the interprocessor communication network, we have chosen to implement a circuit-switched, multistage, indirect network called Transit. These terms are defined in Chapter 2. The Transit network is based on the bidelta network topology, with special network and crossbar switch design features to enhance performance and fault-tolerance. The goals for the Transit network are

- *Low-latency* communication
- *High-bandwidth* communication
- *Fault-tolerant* communication

By low-latency communication, we mean interprocessor message times which are comparable to today's main-memory access times. While some parallel computation paradigms, such as dataflow, claim to be able to mask message latency with parallelism, we would like the option of very high speed message delivery. The initial Transit network is designed for a constant three pipeline stage delay, with very high probability of successful message delivery on the first transmission attempt [DeHon 90b].

The network should support data transfers which match the input-output requirements of the processors. The goal for the Transit network is to support $100 Mbytes/sec$ data transfers on each network port. Each processor node will have four network ports assigned to it on which it can transmit one $100 Mbyte/sec$ data stream and receive two such streams simultaneously.

The third design goal is fault-tolerance. We want our network to be able to provide full connectivity for all processing nodes in the event of multiple failures of routing chips, wires, or connectors. Fault tolerance in the Transit network is achieved through a combination of design

strategies [DeHon 90a].

- *Hardware Level*

  As detailed in the following chapters, the basic routing switch component, the RN1 chip, can operate as a dilated crossbar [Kruskal 86] which provides redundancy in routing connections. The switch can choose one of two alternate paths to route a connection, based on an internal pseudorandom number generator. This provides the basis for *path dilation* in the network, a technique to improve the routing performance and fault tolerance of the system.

- *Network Topology Level*

  A Transit network is a member of a class of logically equivalent wiring topologies which provide inherent redundancy in the choice of paths through the network from a source to a destination. Research into randomized routing of the redundant paths [Leighton 89] through these networks has shown that remarkably consistent performance is achievable, in spite of the presence of faulty chips or wiring. The RN1 chip is designed to support the Transit network topology.

- *Layered Protocol Level*

  The Transit network provides what is called *unreliable* message delivery. The pejorative connotation of this term is somewhat deceptive. In the network literature [Tanenbaum 81], any protocol which gives the sender responsibility for message delivery is termed unreliable. The network makes a best-try attempt to deliver data to its destination. The network provides information to the sender on the status of a connection, but if a connection is blocked, it is up to the sender to retry transmission or choose another destination. This is the same approach used in the packet switched Internet Protocol [Comer 88]. Of course, a reliable packet or stream layer can be built on top of this protocol just as a reliable protocol such as TCP/IP is built on an unreliable medium such as ethernet. The robustness of the ARPA internet protocols attests to the fact that an unreliable network level protocol does not imply unreliable end-to-end data communications.

  While some routing networks claim to provide reliable message delivery in hardware, the lower overhead of the simple unreliable protocol supported by our network hardware helps

us achieve very high speed connection setup and data transmission. Also, the experience of the ARPA Internet has shown that the depending on guaranteed message delivery by the underlying network hardware has proved to be an expensive and error-prone approach.

Chapter 2

A Review Of Interprocessor Communication Networks

The family of multiprocessor computers we are building [DeHon 90b] all have the common need for processor nodes to communicate through a low-latency high-bandwidth network. The basic configuration as shown in Figure 2-1 consists of a number of processor nodes with local cache and memories and interfaces to the routing switch.

The routing switch should ideally be able to make a connection from any input terminal to any output terminal. Since each processor node can handle only a small number of input connections at a time, it is pointless to give the network the capability of routing all inputs to a single output. The inverse capability, broadcasting data from a single processor to all others, might be useful in some cases; however, interprocessor broadcast communication has some serious pitfalls. Consider the case where a sender node broadcasts a message to every other node in the machine and then wishes to get positive ackowledgement from all receiving nodes. An extreme traffic jam of incoming messages will flood the source node and all paths in the network leading to it. This is one example of a *synchronization problem* [Tanenbaum 87], a very serious issue in multiprocessor designs. In general, problems will arise when several processors are competing asynchronously for a single resource.

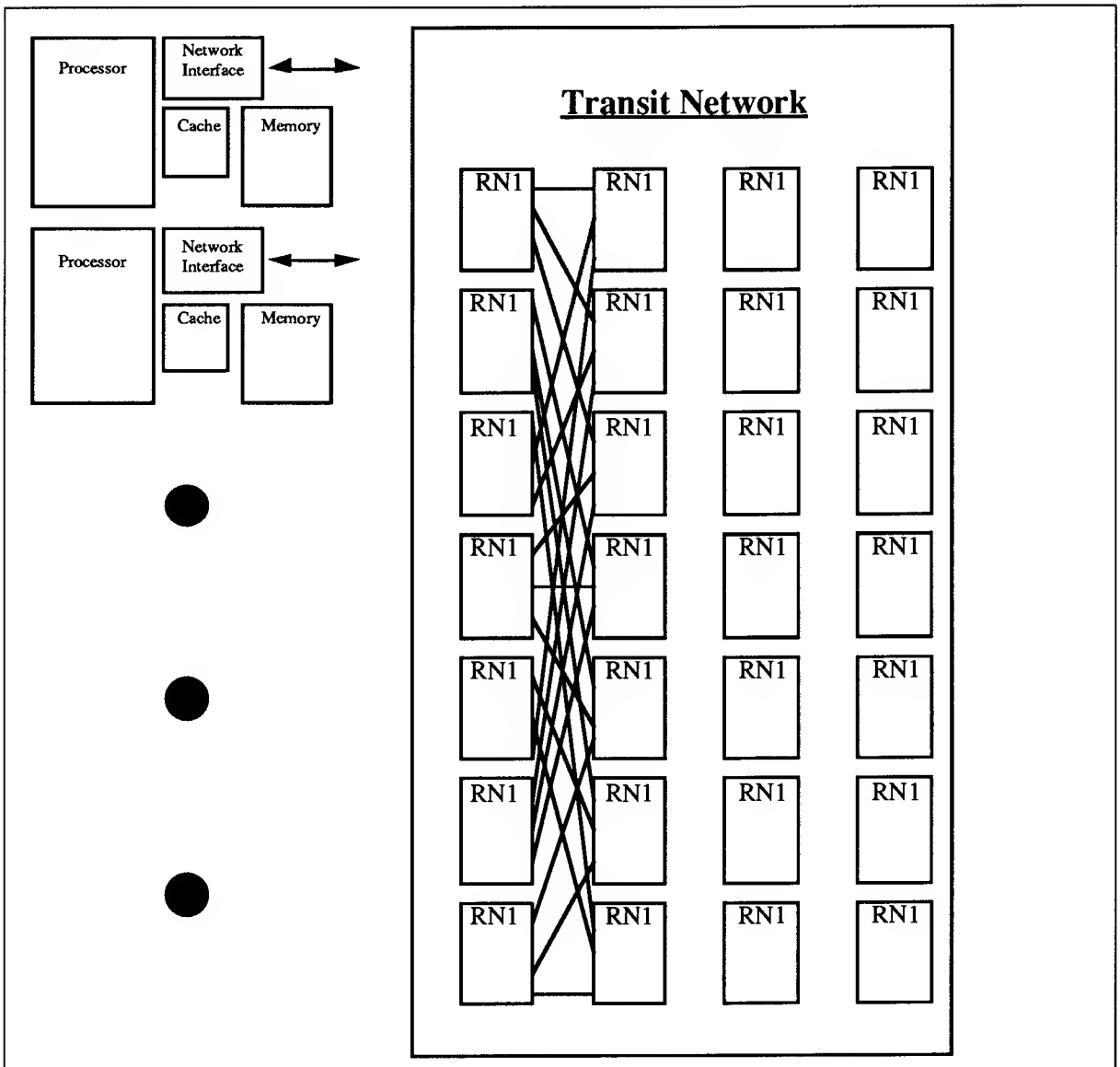


Figure 2-1: A small Transit machine; processors with memory/network interfaces to a routing switch implemented by a Transit routing network.

2.1 The Crossbar Switch

We define an $n \times m$ *crossbar switch* to be a component with n *input ports* and m *output ports*, which can establish a connection between any input port and any output port. The term *crossbar* comes from the old electromechanical telephone switching equipment, which had crossed rows and columns of metal bars for the input and output ports, which could be connected by electromagnetic mechanical contacts. Figure 2-2 shows an example of a 2×2 crossbar switch.

A crossbar switch can be characterized by its *radix*, which defines the number of choices of output directions the part has. Note that the radix of a crossbar is not always equal to the number of output ports. An additional parameter, the *dilation*, characterizes how many physical channels there are in each logical direction. For an $n \times m$ crossbar, $dilation \times radix = m$. The dilation can be thought of as a measure of the redundancy of the logical channels. Figure 2-3 shows a 4×2 crossbar switch with dilation 2, with the output ports grouped together into logically equivalent pairs. This switch can be thought of as switching data between two logical directions. A dilation greater than one indicates that a logical channel can support several simultaneous data transmissions on that channel.

In general, in a self-routing network (Section 2.2.7) with switch nodes of dilation greater than one, the decision of which equivalent physical output channel to use is made by the individual routing elements, with perhaps some feedback from the network as to which paths are more likely to succeed. When a dilated crossbar is faced with the choice of several available channels for an output, it is free to choose which available physical channel in the logically equivalent set will be taken.

2.2 A Routing Switch Made From A Single Crossbar

A routing switch of arbitrary size, which can establish connections from any input to any output, can be implemented using a single large crossbar switch. A true crossbar switch has the unfortunate property that the number of active elements (crosspoints) scales as the square of the number of inputs to the switch. The usage of resources in large crossbars is also very wasteful; for a 1000×1000 crossbar, there will be 1,000,000 crosspoint elements. Only at

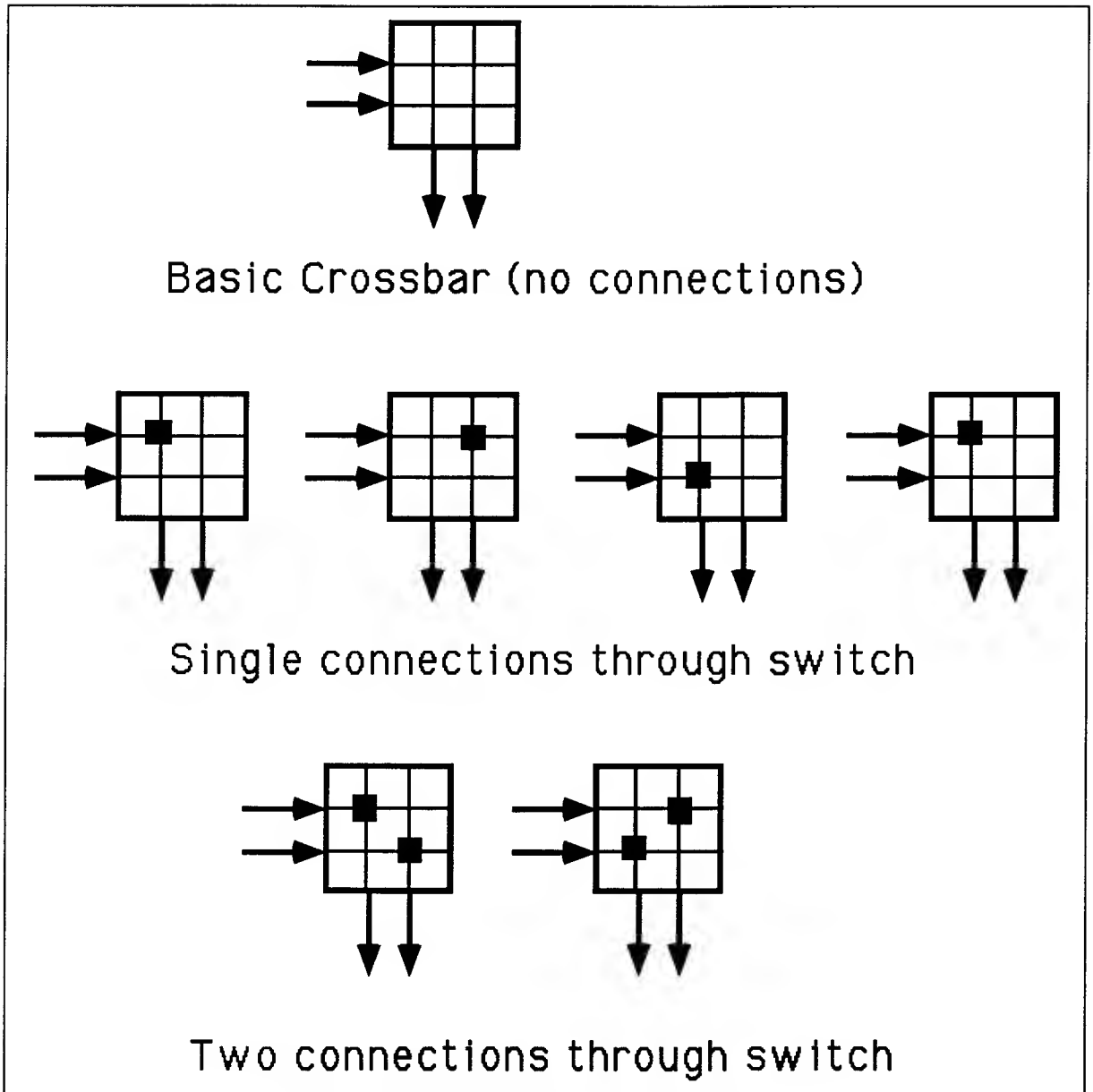


Figure 2-2: A simple 2 x 2 crossbar switch.

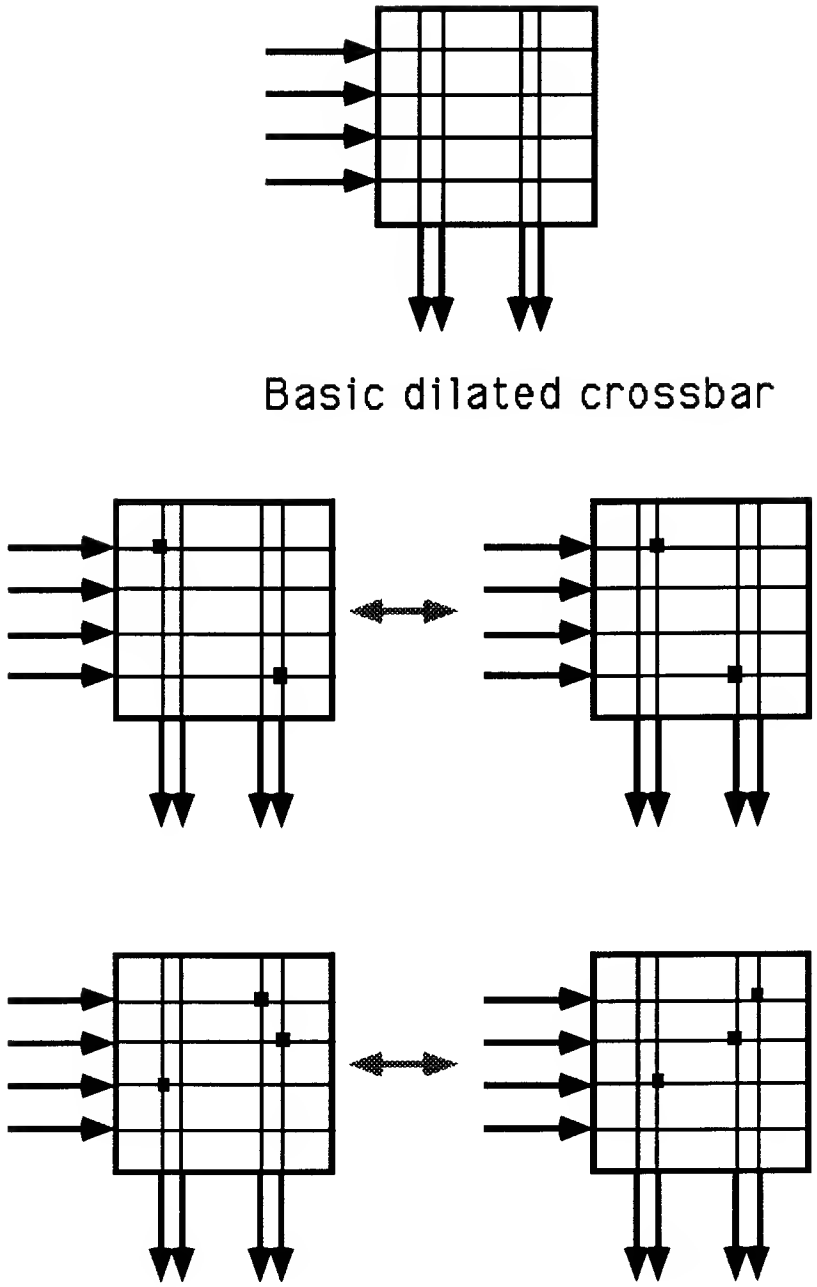


Figure 2-3: A 2 × 2 crossbar switch with dilation 2.

most 1/1000 of the crosspoints in the switch can be active at once, and the remaining 999,000 crosspoint elements will be idle.

2.2.1 Multistage Routing Networks

While the crossbar provides the simplest model of a routing switch, more hardware efficient routing networks can be built by dividing the routing hardware into a distributed or multi-layer structure. The tradeoff is that now several stages of switching are required to route data from the inputs to the outputs of the network. The wide range of possible multistage routing networks can be divided into several broad classes. Table 2.2.1 shows some of the major categories of routing topologies, and the machines which use such networks. Note that in the best cases, the latencies of these networks is logarithmic in the number of network ports, versus constant time for a full crossbar implementation.

Network Type	Example Topology	Complexity	Latency	Processor
Full Crossbar	Full Crossbar	N^2	$O(1)$	Cray Y-MP
Grid/Mesh	Grid	N	$O(\sqrt{N})$	J-Machine
Logarithmic, Direct	Hypercube	$N \log_2 N$	$O(\log_2 N)$	Connection Machine
Logarithmic, Indirect	Omega	$N \log_r N$	$\log_r N$	NYU Ultra
Logarithmic, Indirect	Beneš	$4N \log_r N$	$2\log_r N - 1$	GF11

Table 2.1: Categories Of Networks and Representative Architectures

The networks which I will concentrate on in the following sections are the *multistage shuffle-exchange networks* [Gottlieb 89]. A large group of network topologies are included in this category including *banyan*, *Beneš*, *delta*, *omega*, *butterfly*, and many others. The basic routing algorithms of these networks is the same, with a progression of messages advancing strictly forward through the routing network, and depth of the network proportional to the logarithm of the number of destination nodes. The differences between these networks have to do with geometric layout, blocking performance, component count and network depth.

2.2.2 Direct vs. Indirect Networks

The terms *direct network* and *indirect network* are used to describe the relative topological placement of processors and routing elements in the network. The direct networks intersperse processing elements with routing elements. The *mesh* or *grid* topology places routing switches

and processors at vertices in a two or three dimensional grid. Latency in a grid or mesh of n processors is proportional to $\sqrt{n}$ or $\sqrt[3]{n}$. Data are routed by passing them from point to point in the grid. The binary hypercube places processors and routers on the vertices of a higher dimensional cube. Latency in an n node hypercube is proportional to $\log_r[n]$, where r is the degree of the vertices. One example of a commericial hypercube machine is the Thinking Machines Corporation Connection Machine.

An indirect network separates the routing network from the processors. In an indirect network, the latency for connections through the network tends to be more uniform, since there is usually a constant number of stages between inputs and outputs of the network.

2.2.3 Non-Blocking Circuit Switched Networks

The simplest multistage shuffle-exchange network topologies, such as the *onega* network, have the property that there is one and only one path from a given input port to a given output port. This creates the unfortunate situation that there are many possible states of the switches such that a circuit path from a particular input to an output is blocked by another circuit path. Thus, it is not possible to always open a connection reliably from any port to another. Figure 2-4 shows an example of a blocked path in an 8 port network. There exists an open connection from input port 1 to output port 1, but there is then no way for input port 5 to connect to output port 0, because both path need to use the single upward physcial channel of the crossbar switch in the middle stage of the network.

2.2.4 Clos Networks

It is possible to design a multistage circuit switched network in which all permutations of sender to receiver connections can be made simultaneously, with no blocked paths. One such multistage network is the Clos network. A non-blocking Clos network, shown in Figure 2-5 is a $n \times m$ input three stage indirect network whose first stage is made from r $n \times m$ crossbars, and whose middle stage is built from m $r \times r$ crossbars. It can be shown [Benes 65] that when $m \geq 2n - 1$ and when using a simple routing heuristic, that all possible permutations of input to output connections can be made. For large networks, the size of the needed crossbars in Clos network clearly makes it impractical. It also requires a "omniscient" controller to configure the

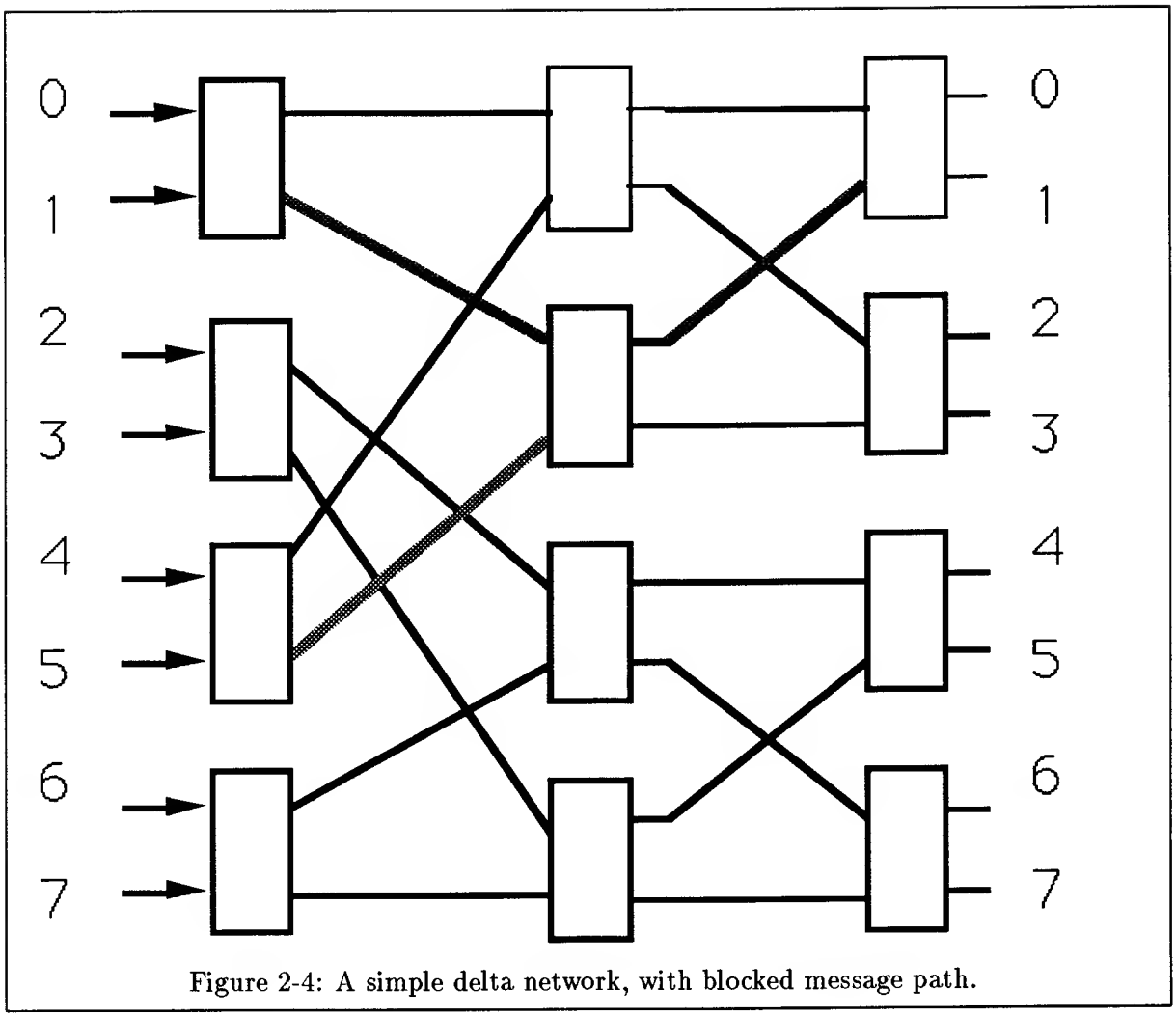


Figure 2-4: A simple delta network, with blocked message path.

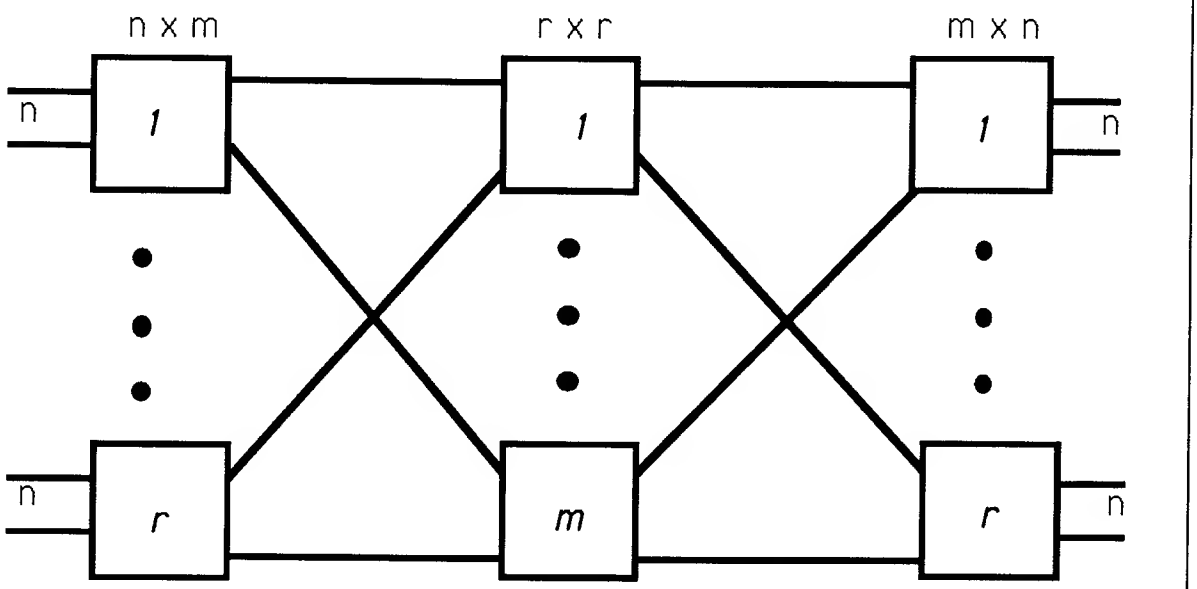


Figure 2-5: A Clos non-blocking network.

switches for a route based on global knowledge of the state of the entire switch.

2.2.5 Beneš Networks

Beneš actually defines two kinds of non-blocking behavior. *Non-blocking in the wide sense* refers to a network for which any possible set of routing switch configurations for a set of connections through the network is possible. *Non-blocking in the strict sense* refers to a network along with a set of rules that if followed carefully result in routing of all messages such that no blocking occurs. Beneš points out that practical networks which are non-blocking in the strict sense have not been found. The Clos network is an example of a network which is non-blocking in the wide sense.

A practical non-blocking network can be created with a recursive formulation of a network built from the Clos network [Hui 90]. For a Beneš network with $N = 2^n$ ports, the construction shown in Figure 2-6 results in a non-blocking network which has $2\log_2 N - 1$ stages and $4N\log_2 N$ crossbars.

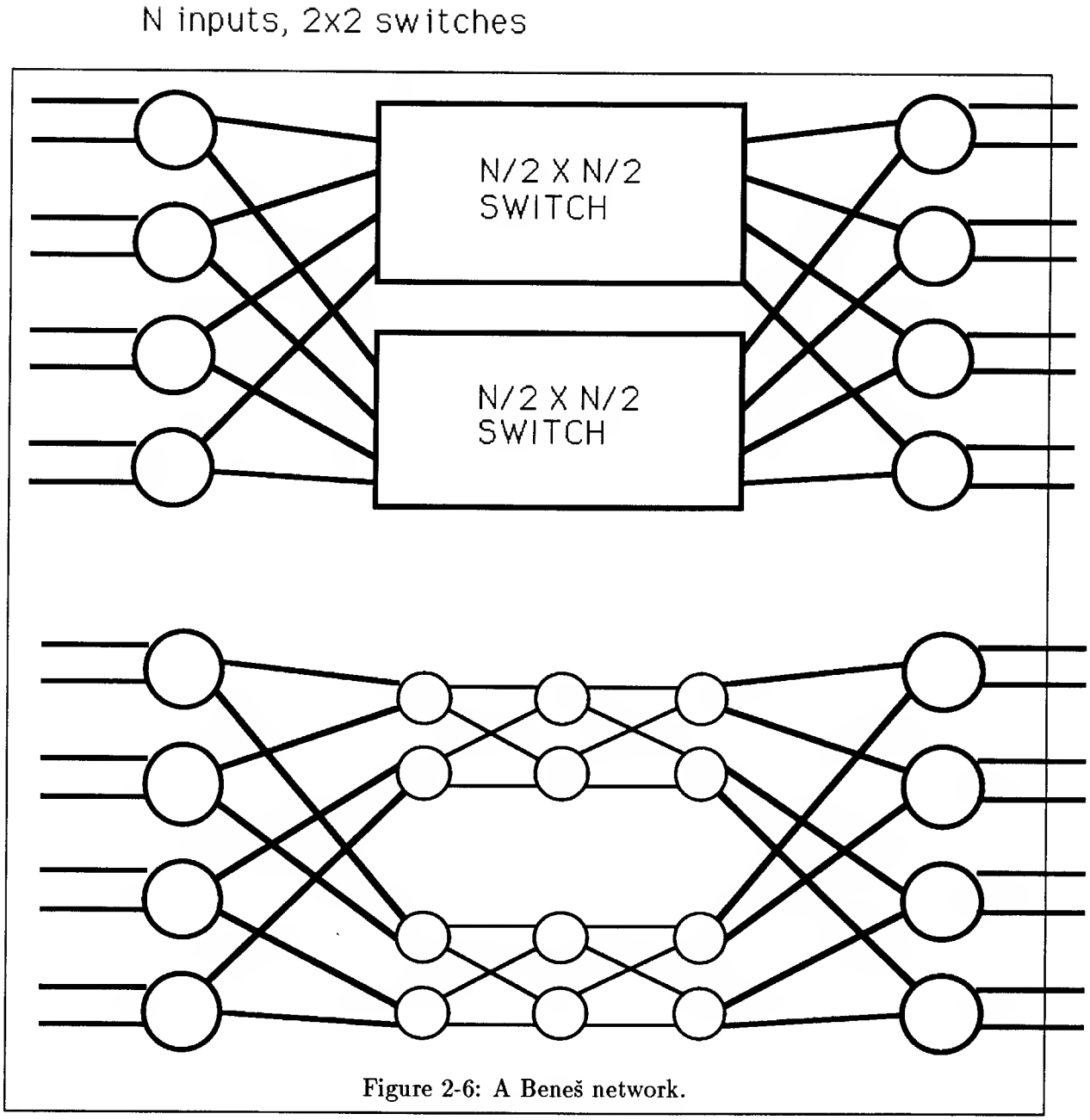


Figure 2-6: A Beneš network.

2.2.6 Packet and Circuit Switched Networks

There seems to be no way to build an efficient non-blocking multistage network with less than $2\log N$ latency. And even if there was, it does not help in the case where multiple messages really want to go to exactly the same destination, i.e., the routing problem is not a simple permutation. There are two alternative approaches to working around the blocking problem in multistage networks. One approach is *packet switching*, where hardware is added to the network which can buffer blocked messages until the resources needed to route them become available. Another approach is to stick with circuit switching and move the responsibility for retrying blocked connections from the network to the sender.

In packet switching, the sender composes a complete packet of data, along with a destination address, and hands it to the network. The network is usually designed to guarantee that the packet will be delivered eventually to its destination and not lost or corrupted. Accomplishing this goal can be much more complex than it first appears; consider the case where a router switch fails in the middle of forwarding a section of a message. The network must have hardware to deduce that data has been lost in transit, and reproduce the lost data somehow.

In a circuit switched network, on the other hand, the sender requests a connection to a destination node, and the network opens a "virtual circuit" through the routing switches. The sender can then send data through this circuit for as long as it wants after which the circuit is closed down. If a path is blocked, the sender is notified and asked to try again later.

An analogy can be made between a packet switched network and the post-office, and between a circuit switched network and the telephone network. With a packet-switched network, the sender puts a message in an envelope, writes the address on the outside, and gives it to the post-office. The post-office eventually delivers it. In a circuit switched network, sending a message is like making a telephone call. The sender picks up the phone, dials the number, and then waits for a connection. If the line is busy, the sender hangs up and tries again later. If a connection is granted, the sender can transmit data to the destination. The connection is held open as long as the sender is transmitting data.

There is even a kind of hybrid of packet-switching and circuit switching. [Dally 87] describes wormhole routing which combines a kind of wormlike packet switched message delivery. A message is divided into *flits*, where each flit is the smallest unit of data that can be sent across

a network edge in one cycle. A message snakes its way through the network, holding open a circuit the length of its flits, and possibly being buffered or diverted in the network. The wormhole router has the property of guaranteed delivery by a network with no component or wiring faults.

2.2.7 Self Routing Networks

One very desirable feature to have in a multiprocessor communication network is the ability for messages or connections to be routed through the switches based only on local knowledge available at each switch. The telephone companies, in contrast, have switches which are controlled by a global switching program, which uses routing algorithms based on the state of the entire network. This is acceptable when the setup time for a connection or message is short compared to the duration taken by the transmission of data in the connection. For a telephone call, the setup time may take milliseconds, and the call may last for minutes. In a multiprocessor system, however, the normal mode of operation will be huge numbers of short message transactions which represent memory references. It is impractical to build a controller to globally service all of these simultaneous routing requests. It is necessary to have the individual switches in the network route the messages based on the local state of the switch, and perhaps its neighbors. Work has been done by Leighton [Leighton 89] on the distributed propagation of network supervisory information by the switch elements in a multibutterfly, and Chong [Chong 90] on local routing algorithms based on propagation of network state using analog circuit principles.

In a simple undilated multistage shuffle-exchange network the routing algorithm is simple; there *is* only one logical path from any input to a specific output. Each routing switch chip simply looks at a portion of the address field of the message and switches the connection to the indicated output port. If that output port is busy, the message is blocked. In a packet network, a blocked message must be either rerouted or buffered locally at the switch.

2.3 The Transit Network

The Transit routing network is a multistage indirect shuffle-exchange network. A Transit network is built from dilated crossbar switches except at the final routing stage.[1] Figure 2-7 shows a 16 port Transit network, made from 4×2 *dilation 2* crossbars. The bold lines show all the possible paths from port 6 to port 16.

The choice of a high dimensional indirect network *vs.* a direct network was made partly because of system packaging issues. We wish to get the lowest latency message delivery possible, and are willing to use as much wire (in the form of multilayer printed circuit boards) as we can, within practical engineering limits, to achieve this goal. The simplicity of routing in the delta network makes it that much easier to achieve high performance in our routing switch components. The simple delta network [Patel 81] has serious problems, from both a fault-tolerance and traffic congestion perspective; since there is only one unique path from any input to any output, a single switch node failure or blocked path will prevent connection between that input-output pair. This problem is addressed by the dilated network topology.

[Kruskal 86] defines a d-dilation of a banyan network to be the network obtained by replacing each interstage channel in the original network by d channels. A message entering a switch may exit using any of the d channels going to the desired successor switch at the next stage. The Transit network topology is similar in theory to the class of d-dilated banyan networks since the dilated crossbar switches are used. But the wiring pattern of the d-dilated banyan networks simply replaces each network edge with a dilated edge. The Transit network splits the dilated channels to make sure that they run to different physical routing switches in order to improve fault tolerance.

It should be pointed out that the actual choice of next-stage switch node destinations for dilated channels has a large impact on the tolerance of the network to internal switch failures. [DeHon 90a] describes an analysis of how network reliability in the presence of component failures is influenced by the wiring destinations of dilated network paths. This work is based in part on the ideas put forth by Leighton [Leighton 89] on fault tolerant routing in the multibutterfly network.

[1] [DeHon 90a] explains why, from fault-tolerance considerations, the final stage of the Transit network should be composed from simple dilation-1 crossbar switches.

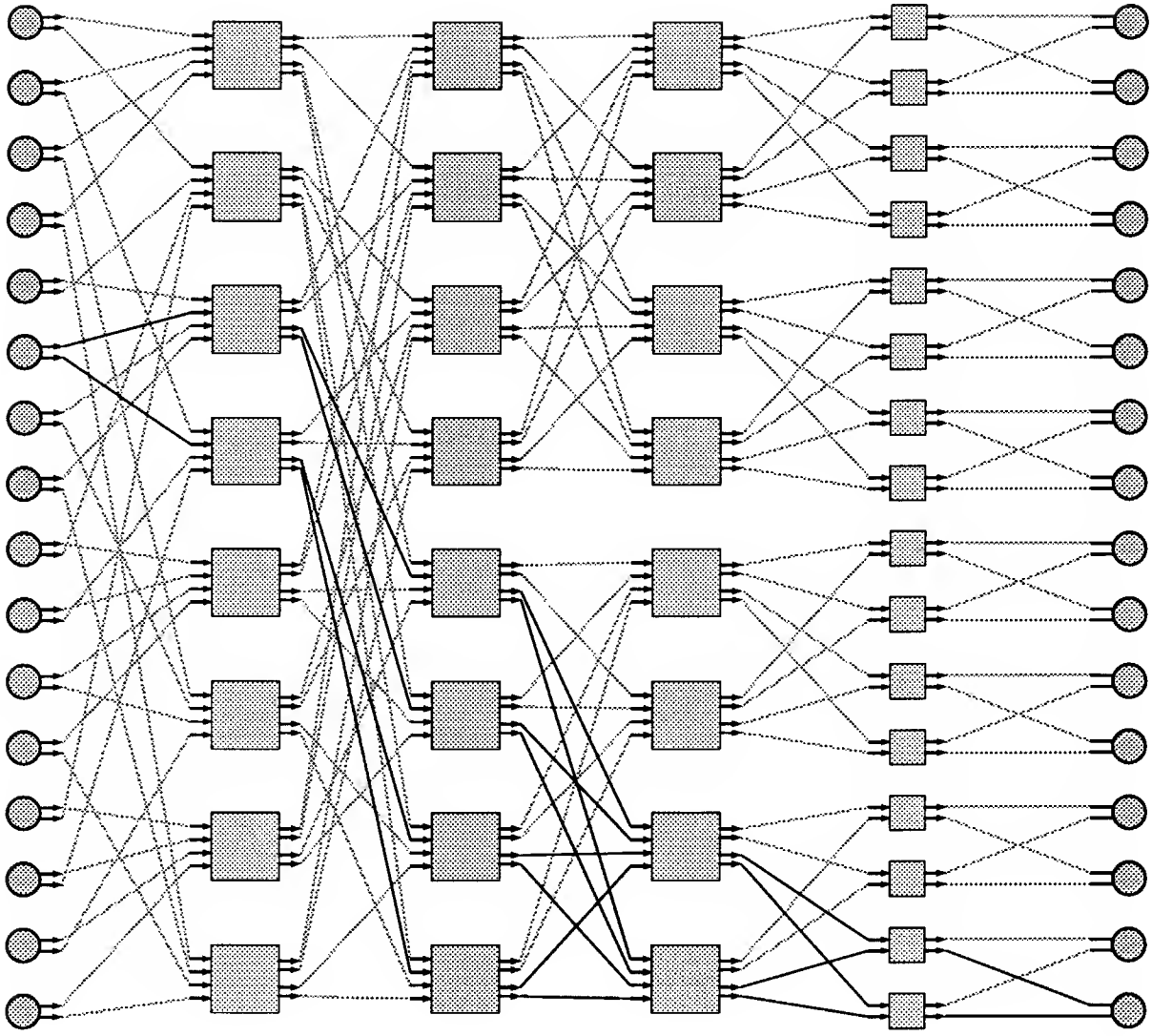


Figure 2-7: A 16 port Transit Network composed of 4 x 2 parallel crossbars.

2.3.1 The RN1 Chip

The RN1 chip, described in detail in the next chapter, is an 8×4 *dilation 2* crossbar switch component with byte wide data channels. It can also be configured as a two independent 4×4 *dilation 1* crossbars in the same physical package.

The Transit network to be used for the MBTA project is composed solely of RN1 chips. The network is organized as n stages of radix r routing chips, where the number of stages equals the logarithm, base r, of the desired number of processor nodes. The primary interface to the network takes place at the input ports of the first stage of routing chips. Connections are dynamically constructed through the network to reach the output ports of the final stage of routing chips. These output ports connect to their associated processor nodes. Each processor node has access to two input ports and two output ports. To keep network loading below fifty percent, we can restrict a processor node to using only one of its two network input access ports at a time. The processor node can still attend to two simultaneously incoming requests on its network output ports.

The Transit network depends on the dilation property of the routing components to provide good routing performance and fault tolerance. The basic mechanism depends on a pseudorandom number generator built into each RN1 chip. When a new connection is being routed, if an RN1 chip has two uncomitted output channels in the desired logical direction, one is chosen at random. This serves to expand the number of alternative paths through the network at each stage. This path expansion helps reduce hot spots in the network due to deterministic communication paths. The randomized routing also makes it likely that an attempted route through the network which fails due to a faulty wire or chip in the path will be routed through a different path when the sender tries to reinitiate the connection.

[Gottlieb 89] makes a distinction between *message-switched* and *circuit-switched* behavior of the network. They define message-switched traffic as using one-way only communication channels, where a reply from a receiving processor node to a transmitting node must be routed as a separate connection. Circuit-switched connections are able to send data bidirectionally.

The Transit network has the capability to support either message-switched or circuit-switched traffic, i.e., a transmitting node with an open routed connection can turn the connection around and receive data from the far end without requiring the receiver to open a new

return connection. Our initial goal is a 64 processor node machine, so with a Transit network built from radix 4 RN1 components, we will have network latency of $\log_4 64 = 3$ through the network, and pipeline delay of 6 for turning around a connection.

Chapter 3

The RN1 Parallel Crossbar Chip

3.1 Background

In Chapter 2, we defined an $n \times m$ *crossbar switch* to be a component with n *input ports* and m *output ports*, which can establish a connection between any input port and any output port. The RN1 chip is a self-routing crossbar switch with byte wide channels. It has two operating modes; it can be an 8×4 crossbar with dilation 2, or a pair of independent 4×4 crossbars each with dilation 1. Figure 3-1 shows a diagram of the functional equivalent of the two modes of operation.

Figure 3-2 shows a block diagram of the RN1 part. The RN1 chip has eight forward ports, FDPORT<A:H>, and eight back ports BKPORT<0:3[A,B]>. We called them *forward* and *back* rather than input and output ports, because data can actually be transmitted bidirectionally through a connection. Connections can only be initiated through a forward port, however. Data transmitted from a forward port to a back port is said to be travelling in the forward direction. Data transmitted from a back port to a forward port is said to be travelling in the backward direction. Detailed description of the operation of the chip is given in following chapters.

In the 8x4 mode, RN1 can open a connection from any forward port to any available back port. In the dual independent 4x4 mode, the RN1 acts as two separate 4x4 routers, where the forward ports A,C,E,G can connect only to back ports 0A,1A,2A,3A, and the forward ports B,D,F,H can connect only to back ports 0B,1B,2B,3B.

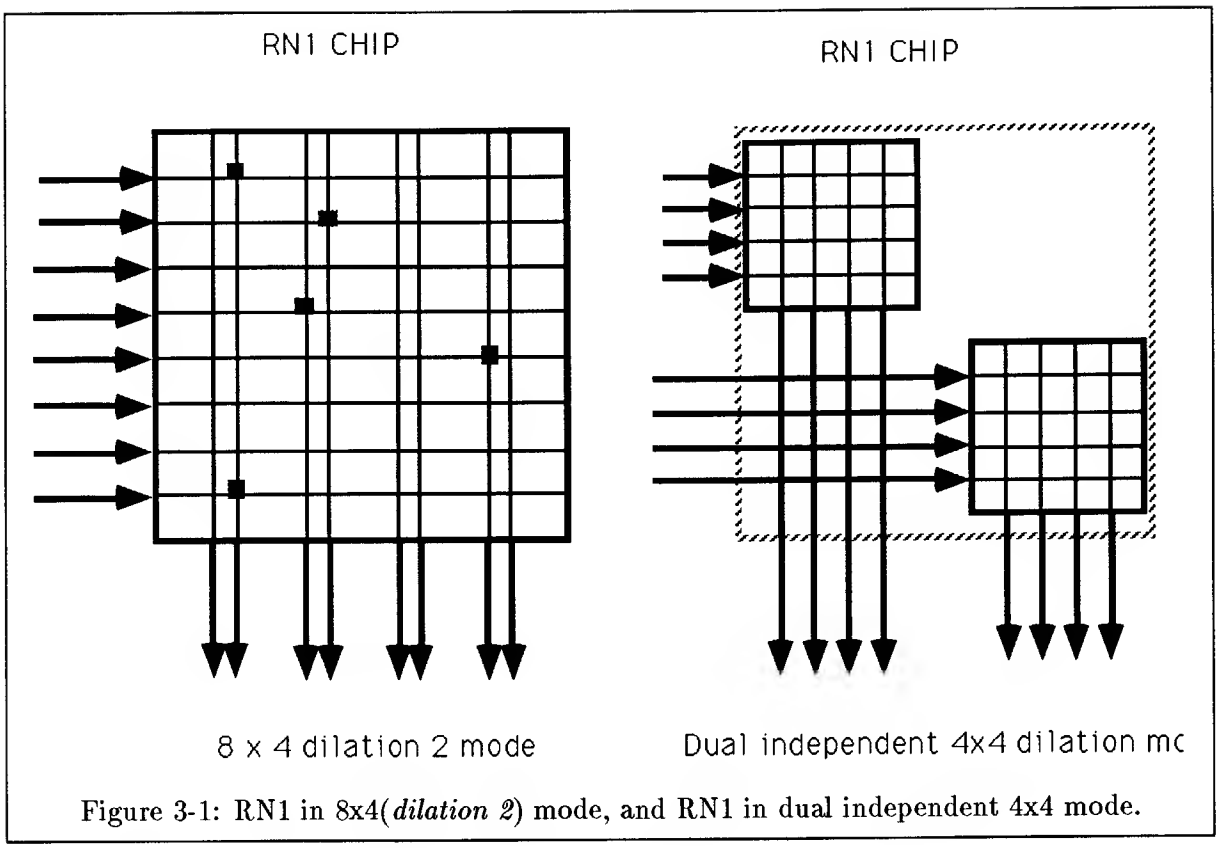


Figure 3-1: RN1 in 8x4(*dilation 2*) mode, and RN1 in dual independent 4x4 mode.

3.1.1 The Need For Dilation

An essential feature of the RN1 chip is its dilated outputs; the ability for it to route a connection to one of two logically equivalent output ports. This mode of operation allows the construction of a routing network with multiple paths from any source to any destination, such as the Transit network. In fact, the number of alternate paths expands exponentially up to the middle of the routing network [DeHon 90a]. This is in contrast to the simple crossbar switch of dilation 1 as used in a basic omega or butterfly network.

For a network of given size, as measured by the number of ports, number of routing chips, and number of wires, a single Transit network has better routing performance under load than a network built with the same number of parts as two independent omega networks. Dilation is also of great importance for fault tolerance in the network. The Transit network can have a remarkable number of chips removed before any processor loses full connectivity with the rest if the network [Egozy 90e].

3.1.2 Connection Protocol

RN1 supports a *circuit switched* connection protocol. This means that once a connection has been opened, an arbitrary amount of data can be streamed through it, and it will be delivered to the destination in order with a fixed latency. There is no fixed packet size, at least not at this level. Higher level protocols can of course impose additional constraints on the data format. A connection through the Transit network consists of a path through a set of RN1 chips with sequential interstage open connections. The RN1 chip, and thus the Transit network, is *self-routing* because a connection is created through the network with each RN1 switch in the path making a routing decision based on purely local information. This is in contrast to an architecture which requires routing control inputs distinct from the data ports, as is common in telecommunications crossbar switches[Gigabit 1988][Barber 88].

3.2 System Design Issues

The RN1 chip was designed to be a part of a complete computer system. The integrated circuit itself is only a part of the final system, and many aspects of its design were driven by

considerations of how it fit in to the system as a whole. The Transit project goal is to build a practical working system within the bounds of today's technology. Ideally the whole computer system would be fabricated on a single monlithic substrate. The practical size of a chip today is about 1.5 cm square, and contains only a few million transistors. The systems we want to build are larger, and thus require many chips to be packaged and interconnected, powered, and cooled. The system must be partitioned into modules, and the decisions of how to best partition the system are driven by technology constraints and our imaginations.

One of the overall goals of the Transit project is the design of a physically compact and reliable packaging and interconnect system in which hundreds of processor nodes can communicate with one another quickly. A Transit network supporting 256 processors is composed of 256 routing switch chips, and the inevitable support circuits for clocking and interfacing to the nodes. The design decsions for RN1 were made based on a mix of contributing factors, some of which are detailed in the following sections.

3.3 Chip To Chip Communication Technology

The on-chip worst-case delay for the RN1 is estimated to be between 8 and 12 nsec. The delay going from on-chip to off-chip through a 5 Volt I/O pad is 4-5 nsec. The transit time between chips is limited by the speed of light in wire, in the best case. In reality, reflections of pulses because of impedance mismatch between drivers can increase the settling time. The 5 volt standard CMOS output levels also takes a non-neglible time to swing.

Prof. Knight has described a design for low voltage self-terminated pad driver and reciever [Knight 89b]. This presents a good alternative for future RN1 designs, combining the economy and low power consumption of CMOS with the benefits of ECL-like speed. The proposed 1 Volt pad voltage swing would significantly reduce power consumption of the chip. The design for the 1 Volt self-terminating pad drivers and receivers has been updated by Prof. Knight and Alex Ishii. New pad test circuits are currently under evaluation in our lab.

Another option is to use ECL gate-array technology for the next version of the RN1 chip. The current packaging scheme makes provision for liquid cooling, so the power dissipation is not a serious problem. Motorola has ECL gate-array with series-terminated pad drivers available in several impedance values.

3.4 Packaging

The first technological constraint on system partitioning comes with the number of pads that can be placed on a silicon chip using today's commercial technology. The constraint comes both from the requirement that pads be located at the periphery of chips, a limited perimeter area, and the cost of high pin-count packages. The practical limit is between 300 and 500 pins, with packaging costs rising much faster than linear as additional pins are added. This sets constraints on the choice of radix and dilation of our routing switch. If we commit to byte wide ports, this leaves us a practical limit of sixteen ports on a chip. The reader might wonder at that number, since adding up the port pins plus control signals gives a total of only around 160 pins. For even modest performance on a chip of this size, a sizable number of power and ground pads are needed. RN1 uses 80 power and ground pin from the package to the circuit board, and more even more power bond wires from the package to the die.

The interconnection network is a crucial bottleneck in the system design, because of the very large number of wires that must be routed; With a 256 processor system with one network routing stage per board, up to 8,000 wires have to be routed between the routing stages. Since the required number of chips will not all fit on a single circuit board, this means that these thousands of wires must run between circuit boards. In a conventional computer system design, the routing cards would be fitted to a backplane with edge connectors. If we wish to put a four stage network onto four cards, we will need edge connectors with 4,000 pins. Current technology bus connectors allow more like three or four hundred pins on a EuroDIN style connector. Instead, we have decided to dispense with the backplane entirely.

We have taken a unique approach to the board-to-board connection problem, by using our chip packages as both chip-to-board and board-to-board connectors. This is done by our use of custom designed IC packages and connectors[Transit 90]. Our IC packages have 372 contact points which are used to create a board-to-board connector, with the routing chip in between (see Figure 3-3). This allows our machine to be packaged in a dense stack, effectively using all three dimensions for wiring, as opposed to only using two in conventional packaging.

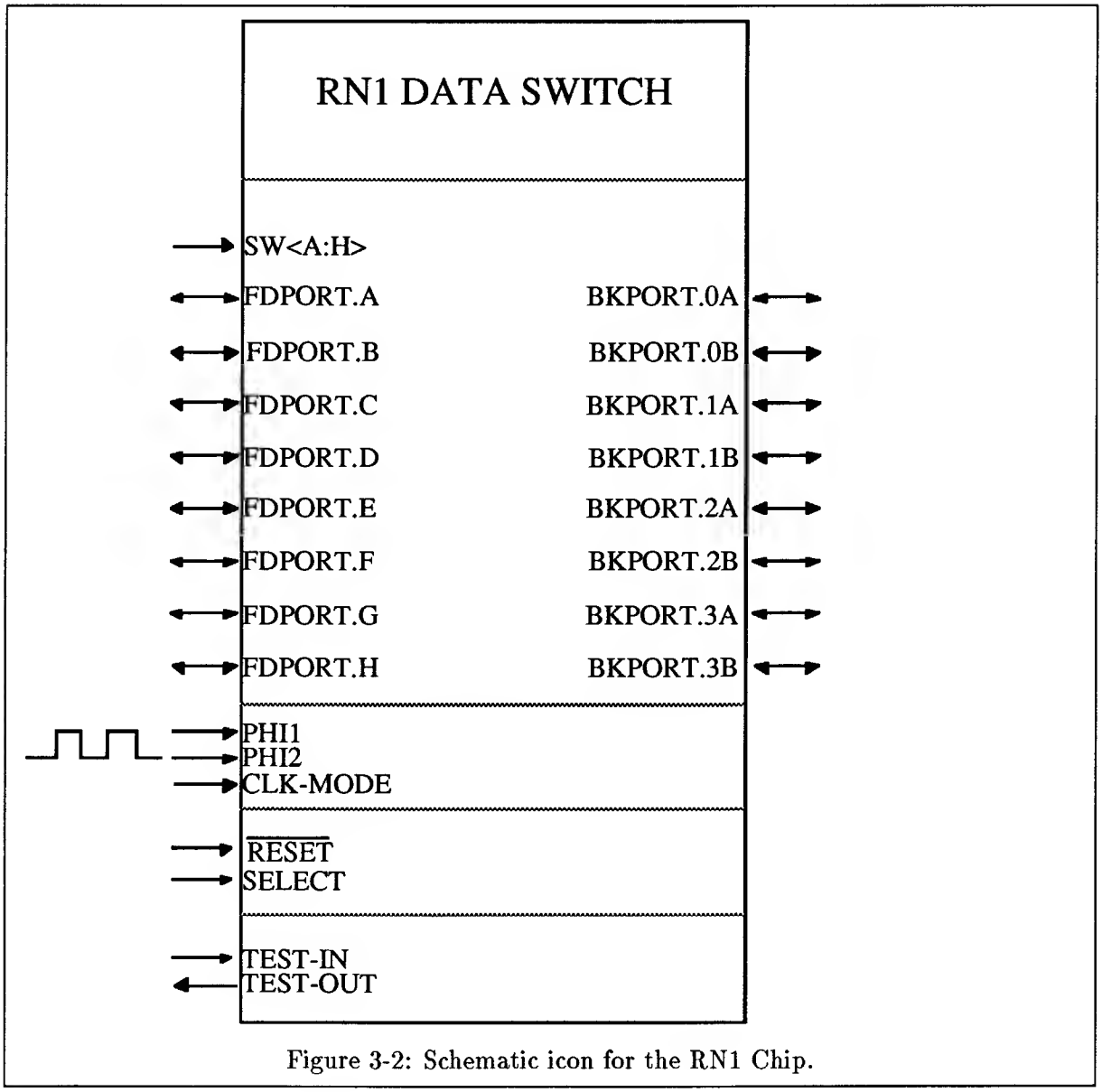
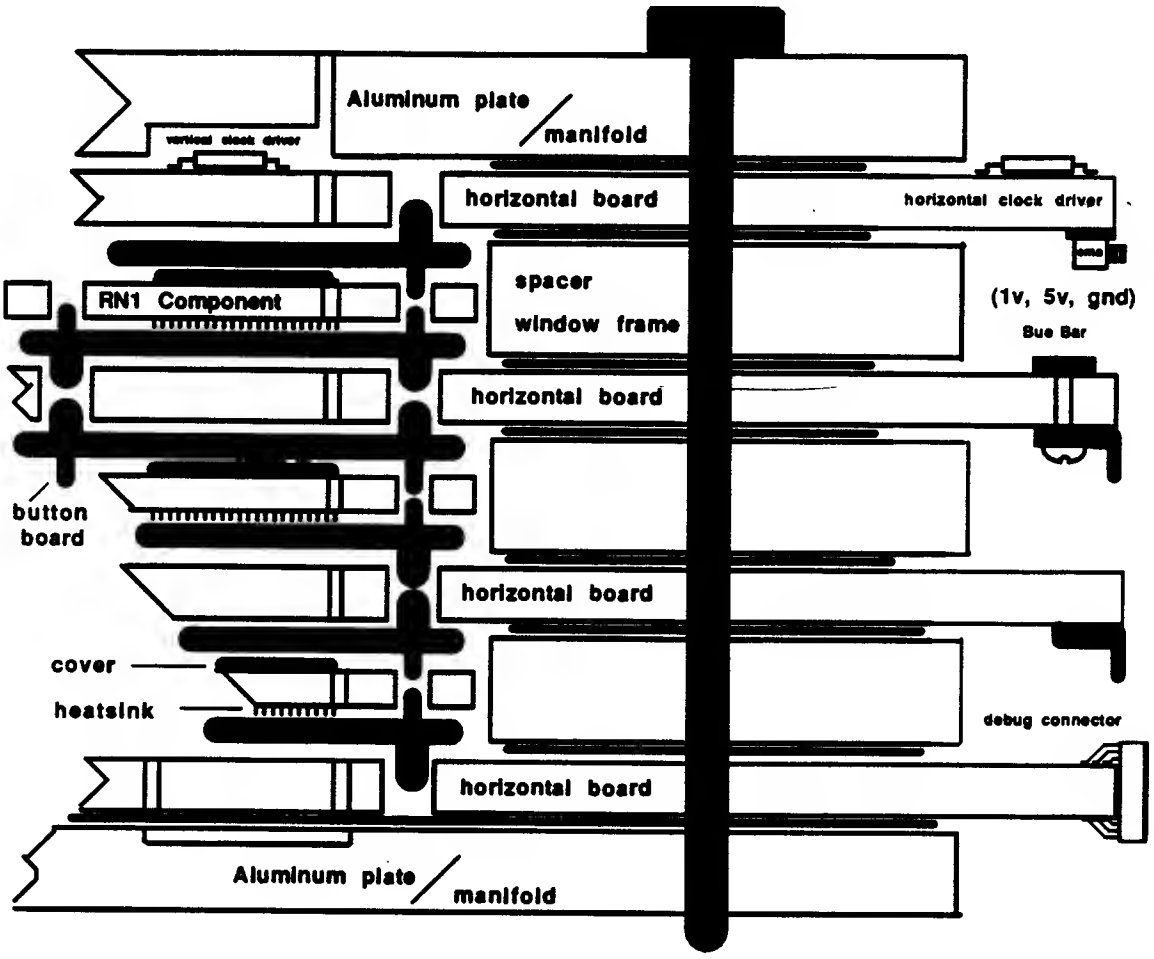


Figure 3-2: Schematic icon for the RN1 Chip.

Stack Cross-Section



(Diagram courtesy of Fred Drenckhahn)

Figure 3-3: The *stack* construction technology developed for Transit. The entire machine is a multilayer sandwich of alternating boards and chips. Board to board connections are by way of the chip packages sandwiched between button connectors. Fluid cooling can be run vertically through the stack.

Chapter 4

Routing Chip Communication Protocol

The Transit network is designed to provide an efficient hardware substrate on which to implement a small but flexible set of primitive network transactions. The essential primitives we wish to implement, as defined in [DeHon 90c], are shown in Table 4.1.

In order to let the Transit network support these primitives efficiently in hardware, the RN1 chip architecture supports a simple protocol which will be described in the following sections.

4.1 Chip To Chip And End To End Network Protocol

When a network is built from RN1 chips, the protocol used by the network interface controllers to talk to the network endpoints is the same protocol as is used internally by the network chips to talk to one another. Thus, it is possible to detail a single command protocol which serves to explain both the external user's end-to-end view of the network, and the internal chip-to-chip

Operation	Description
`read`	Read memory data from a remote destination node
`write`	Write memory data to a remote destination node
`noop`	Open a null connection, used for network testing
`reset`	Issue a hardware reset to a destination node
`rop`	Network operation emulation primitive[1]

Table 4.1: Transit Network Basic Primitive Transactions

COMMAND	ENCODING	MEANING
IDLE	#000000000	This port connection is idle.
ROUTE	#1aabbccdd	Open a connection to address #aabbccdd.
DATA	#1xxxxxxxx	Send this data through an active connection.
TURN	#011111111	Turn control of channel over to receiver, and return status and checksum bytes.
DROP	#000000000	Drop this active connection.
HOLD	#100000000	Temporary pattern to hold connection open during backward turn.

Table 4.2: Byte Encoding of Command Words Understood By RN1

communication protocol.

The RN1 chip is operated by sending sequences of commands and data into its forward ports. Each forward and back port presents a bidirectional nine-bit wide datapath, consisting of eight data bits and one control bit. The ninth bit (MSB) is the control bit. This bit is always high when transmitting data and low when signalling occurs or no data is being transmitted. Data is latched into the port synchronously with the clock, on the the falling edge of the external clock.

4.1.1 Byte Encoding of Command Words

A complete data transaction using one or more cascaded chips in a network, consists of a sequence of command and data words. Table 4.2 summarizes the valid commands which the RN1 supports.

The meaning of the command words is summarized below.

- **IDLE** When a forward port is in its IDLE state, its pins are configured as inputs, and all data presented to it with control bit low is ignored. When a back port is in IDLE state, its pins are configured as outputs, with an idle byte pattern as its output.

  After a global chip reset all of the forward ports are set to be inputs and are said to be internally in an IDLE state. The back ports are set to be outputs with an IDLE pattern (all zeros are driven). All connections in the internal switching matrix are cleared.

- **ROUTE** When talking to an idle forward port, raising the control bit indicates a request to open a connection. The top two bits of the data word indicate which of the four logical output directions to attempt a connection with. If the connection is successful, the routing word is forwarded through the connection during the same cycle. In a multistage network, the 8-bit routing word (and all other data) are rotated left two bits by the interstage wiring, presenting the next two bits for routing in the top of the data byte at the next stage.

- **DROP** The DROP command causes the current connection to be cleared. The DROP command is forwarded through the connection before the connection is dropped. Thus, cascaded chips in the network cause the connection to be ripped down all the way through the network.

- **TURN** The TURN command causes a reversal of the direction of data transmission on the currently open connection. In the forward direction, STATUS and CHECKSUM bytes are returned from chips in the dead time that would normally fill the pipelined path through the network. In the backward direction, HOLD bytes are transmitted for the dead cycle.

The most common type of network transaction we envision is a 'round-trip' connection. A connection is requested to the desired target address. The bytes of a short message are sent through immediately following the routing byte. The connection is then turned, at which point verification is returned, in the form of status and checksum bytes, that a connection was successfully initiated. Reply data, if any, is then transmitted by the node at the far end of the connection.

4.2 Grammar to Describe RN1 Protocol

A valid message sequence can be described by Figure 4-1, where [BYTE]* means a series of zero or more of that type of byte.

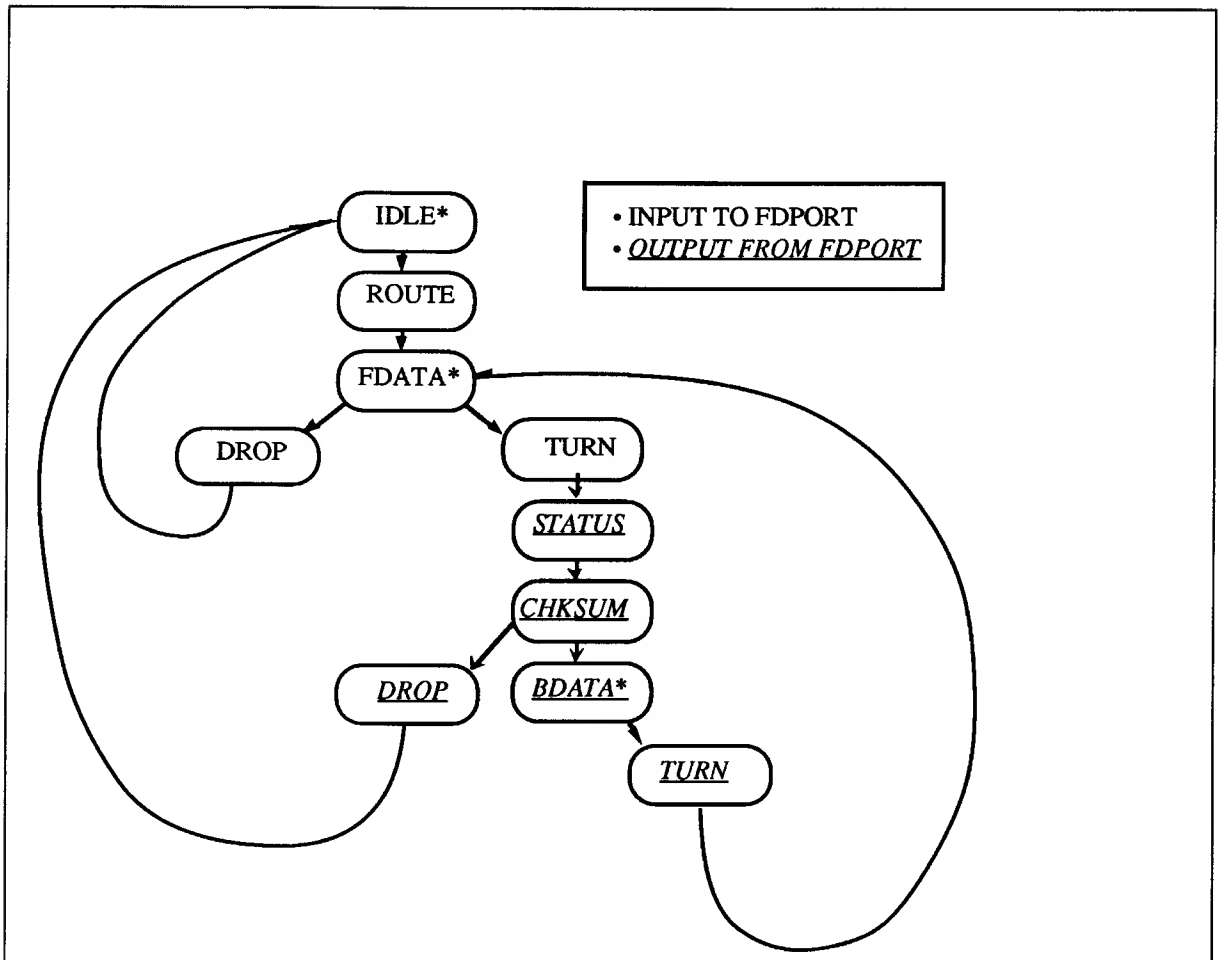


Figure 4-1: RN1 Message Sequence

4.2.1 Opening A Connection

In the IDLE state, the forward ports are watching for a low to high transition of the control bit to indicate the start of a connection request. The host processor should apply idle bytes, consisting of all zero data word and zero control bit, until it is ready to attempt to open a connection. The first non-idle byte received by a forward port is a routing byte.

(MSB)								
CB=1	A1	A0	B1	B0	C1	C0	D1	D0

Table 4.3: Routing Byte Format

A routing byte consists of 8 bits specifying the destination for the message and a 1 in the control bit. The RN1 chip examines the high two data bits (A1,A0) to determine which of the four pairs of output ports will receive the message. Assuming a free output port in that direction is available, the routing byte is passed on through RN1 unchanged during the following clock cycle. Subsequent non-idle (control bit 1) bytes received by the RN1 are forwarded as data bytes. Each data byte is clocked onto the chip on the falling edge of the clock and appears at the outputs, if a connection has been successfully opened, at some point during the remaining period of the clock cycle. The data is thus available on the next clock cycle to the next stage router in the network. The number of data bytes transmitted is determined by higher level protocols, and is not restricted by the RN1 implementation.

In order to cascade more than four levels of RN1 chips, a Forward Port can be configured to take an extra cycle when opening a connection [2]. This extra cycle is used to *swallow* the first ROUTE (nonzero control bit) byte; when swallow is enabled, the first active ROUTE byte is simply discarded. The next byte to be sent is treated as the real routing byte, and is passed along to the back port (when a successful connection was made) in the normal fashion.

4.2.2 Blocked Connection

It is possible that at the time a forward port is trying to route a connection, all back ports in the desired direction are already in use. This creates a blocked path. In a multistage network, the partial path up to this chip will be held open, and subsequent data bytes will be discarded,

[2]To activate the swallow state for port n, the SWn pin should be tied to VDD

until the connection is turned or dropped. The originator of the message will not be able to tell if the path was blocked unless the connection is turned around.

4.2.3 Checksum

A router checksum is computed on the forward data through each switch. The 14 bit value of this router checksum is returned as part of the status byte (see Appendix A) but is otherwise ignored. It is the responsibility of the sender to compare the router checksum with the router checksum of the transmitted data. The router checksum provides a diagnostic to localize faults in the network. Higher level protocols will need a message checksum in the forward direction to assure the destination processor that the received message is intact.

4.2.4 Turning A Connection

At the completion of the forward data transmission a TURN byte consisting of a data word of all ones ($FF) and a zero control bit is transmitted. The receipt of this byte by the RN1 on the input port signals the reversal of the associated data connection. The turn byte code is pipelined out the back port (if a connection presently exists) just like a normal data word. Simultaneously, connection status information and 6 bits of checksum (with control bit 1) are driven back onto the input pins. During the next cycle, another 8 checksum bits (with control bit 1) are driven back to the input pins.

A connection can actually be turned any number of times. The decision whether to drop or turn is always made by the node which is sourcing data into the channel.

4.2.5 Dropping A Connection

Alternately, the connection can be closed down by transmitting a DROP rather than a TURN byte. The DROP byte is pipelined out the back port, just like a normal data word, and the forward and back ports return to the IDLE state on the next cycle.

4.2.6 Turning A Blocked Connection

If the forward port is in a blocked state, then a TURN command has no continuing connection to turn around. In this case, the normal STATUS/CHECKSUM is performed, with the STATUS

byte containing an indication that the connection was blocked at this stage. After STATUS and CHECKSUM are returned, the forward port returns a DROP command in order to shut down the initial partial path through the network, and returns to the IDLE state. [See the Sample Message Transmissions, Section 4.3 for an example of turning a blocked path]

4.2.7 Backward Connection

In the unblocked case, a forward connection which has been turned is now sending data in the 'backward' direction. The connection maintains backward propagation until a DROP or TURN byte is received at the back port. If a DROP is received it is pipelined out the forward port, and the forward and back ports return to the IDLE state on the next cycle.

If a TURN is received from the back port, then on the next cycle, the turn byte is driven out the forward port, while simultaneously, a HOLD pattern (control bit 1, data zeros) is driven out the back port. This holds the connection open during the next cycle, while the turn byte propagates to the previous chip in the network. For an example of this turning of a backward connection, see the third example transmission in the next section.

4.3 Message Examples

The diagrams on the following pages show examples of sample message transmission sessions. The data at the forward and back ports, and their direction of transmission, is shown by the arrows into and out of the center router chip.

4.3.1 Standard Message

Figure 4-2 is a standard message, with two bytes of data sent forward, the connection turned, one byte of data sent back, and the connection dropped from the far end.

4.3.2 Blocked Message

Figure 4-3 is an example of a connection which is blocked when it is opened. All Data bytes are lost, and the status returned indicates that no output connection was grabbed.

Note that the TURN command given to the blocked router returns STATUS and CHECK-

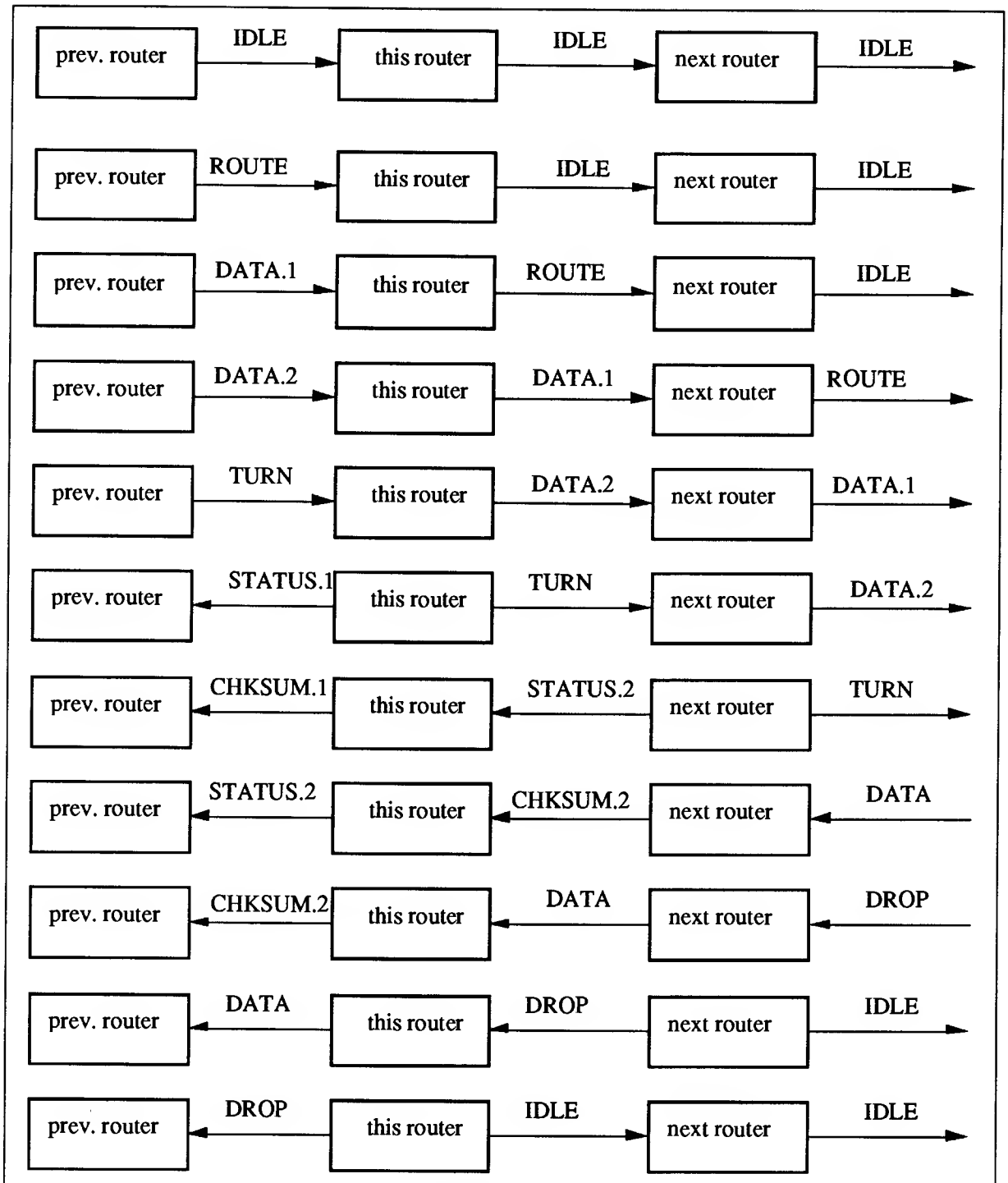


Figure 4-2: Sample Message: Opening A Connection

SUM, just like an unblocked connection. The blocked forward port then issues a DROP command in the backward direction, which will collapse the connection back to the source.

4.3.3 Turning A Backward Connection

Figure 4-4 shows an example of a connection which has already been turned (a *backward* connection) being turned from the back, to become a forward connection again.

The HOLD bytes are distinguished by the suffix *.a*,*.b*,*.c* to show which chip is generating them. They correspond to the STATUS and CHECKSUM bytes in the forward connection turn sequence.

The following diagrams show on each line sequential snapshots in time of the state of a chain of three router chips in a network. The type of data word being transmitted and the direction of data flow is show on the arrows connecting the chips.

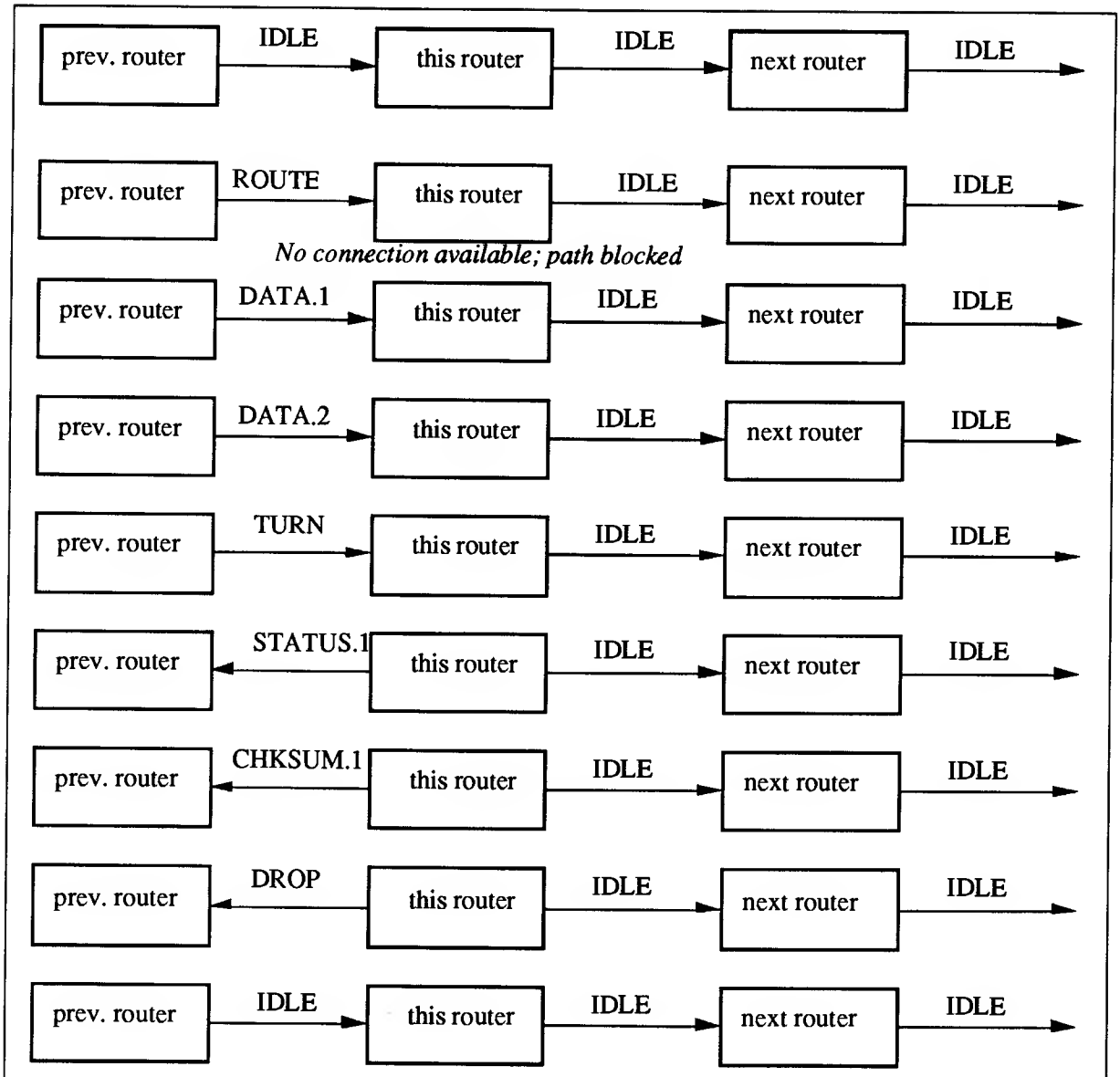


Figure 4-3: Sample Message: A Blocked Connection

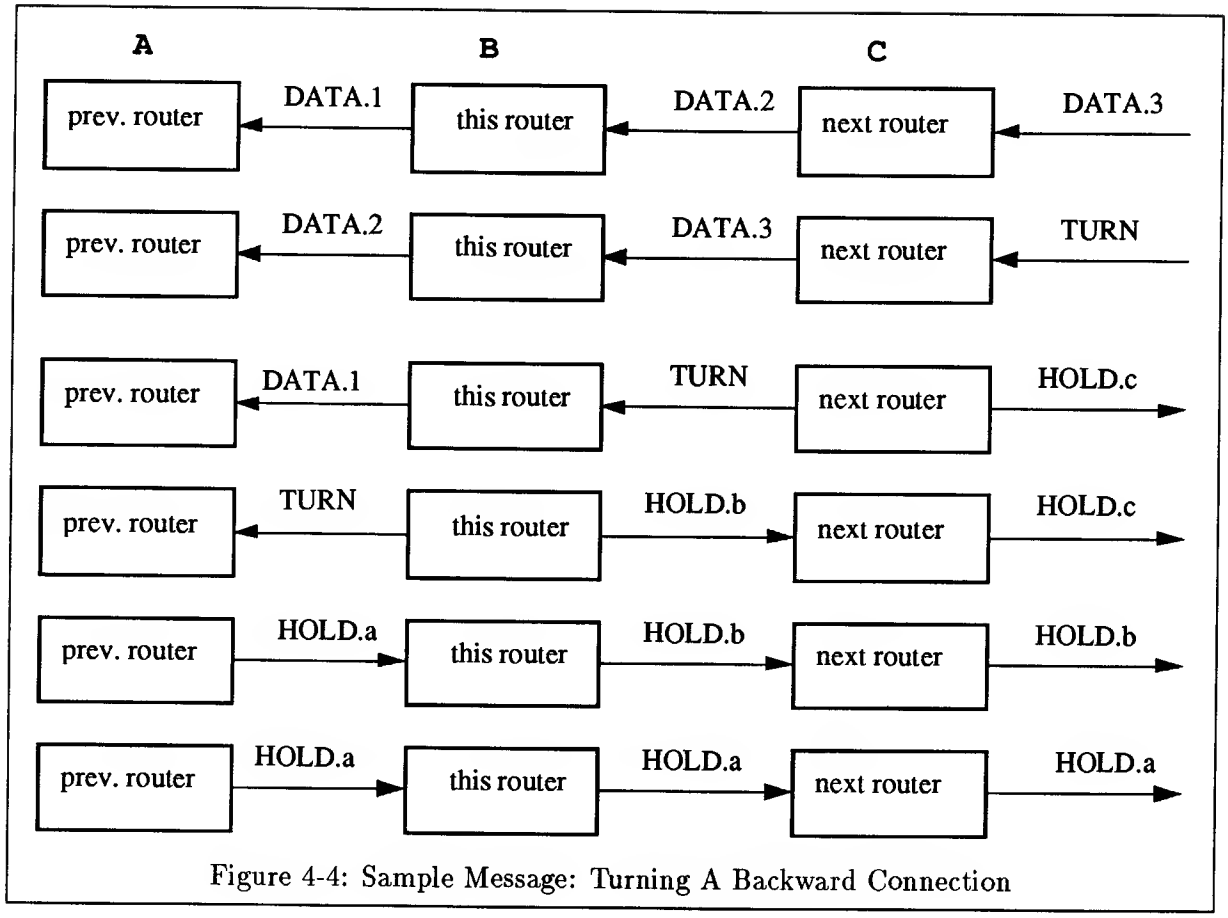


Figure 4-4: Sample Message: Turning A Backward Connection

Chapter 5

Architectural Description of the Chip

5.1 Overview Of The Internal Chip Architecture

Internally, the RN1 chip consists of a central switching fabric, surrounded on four sides by forward-port and back-port state machines. The forward-port busses run horizontally, and the back-port busses run vertically. The central switching fabric is a tiled array of crosspoint modules. A crosspoint module is located at each intersection of a horizontal bus (forward port) and a vertical bus (back port). Because of the intrinsic redundancy built into the routing architecture, each crosspoint is actually a dual structure, serving a pair of back ports. The crosspoint array is shown in Figure 5-1 as an eight row by four column array of these modules, to better indicate the the close relation between the pairs of back ports. Each column represents one of the four logical routing directions, and the two back port busses in each column provide access for up to two connections in that logical direction.

The following sections detail the construction and operation of the forward ports, back ports, crosspoint array, and line control modules. Clocking and pad logic is also discussed.

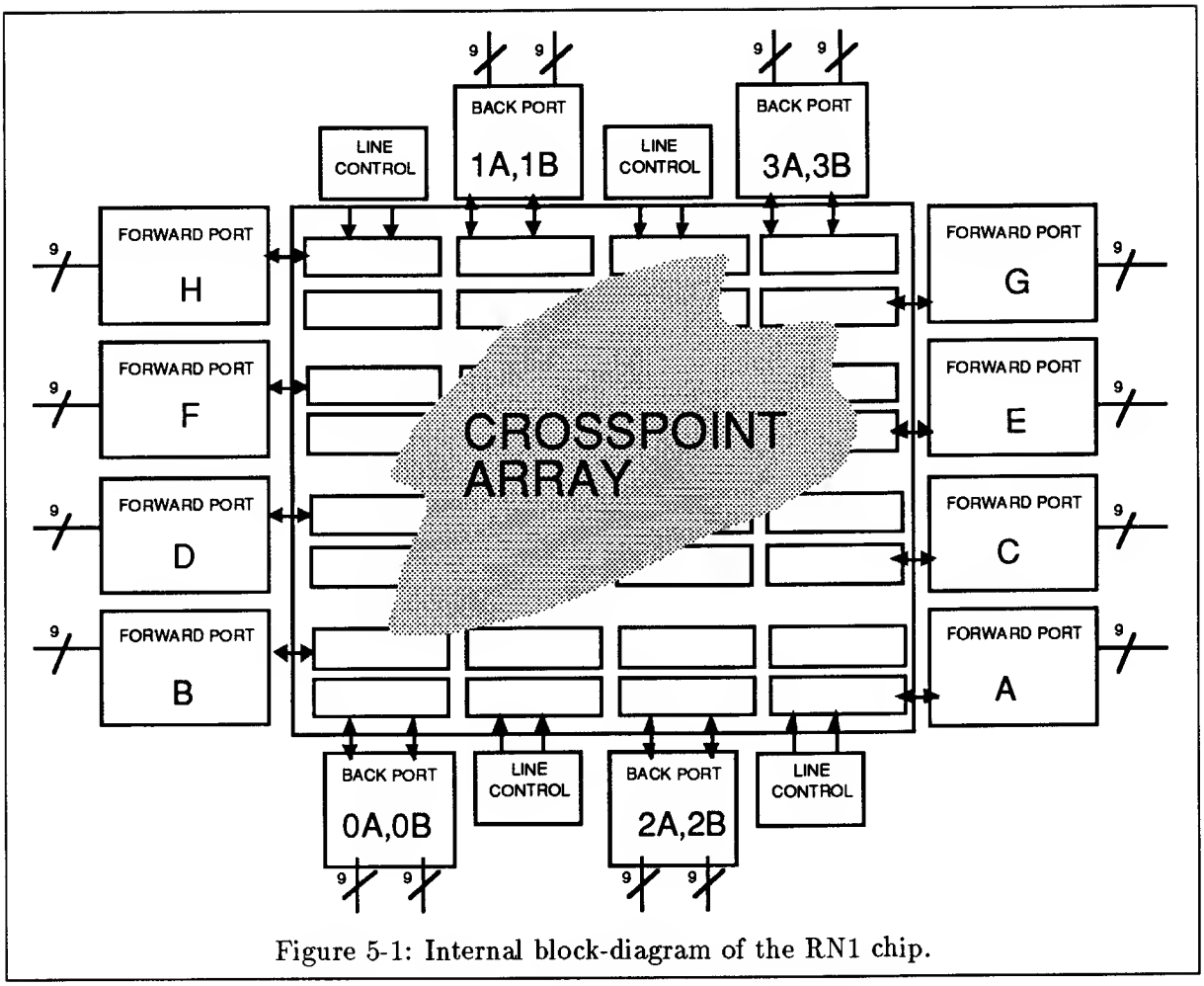


Figure 5-1: Internal block-diagram of the RN1 chip.

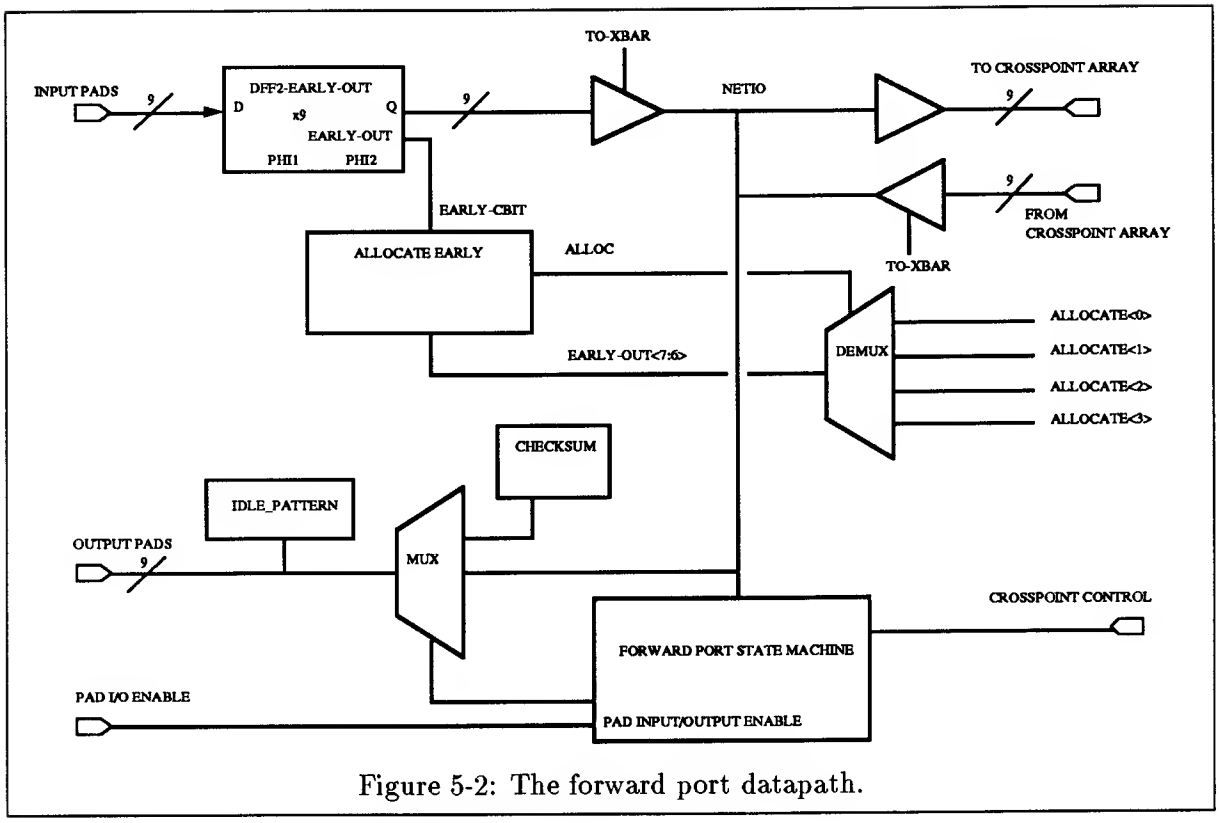


Figure 5-2: The forward port datapath.

5.2 Forward Ports

The eight forward ports have the primary responsibility for decoding and opening new connection requests, and returning connection status when a connection is turned. The basic datapath of a forward port is shown in Figure 5-2

The forward port attempts to open a connection using one of its four `allocate<3:0>` control lines into the crosspoints, as selected by decoding the top two bits of a routing byte. Each `allocate` line is wired directly to one of the four crosspoints on the internal horizontal data bus. Each crosspoint is capable of making a connection to one of two back port busses in the four logical directions. Crosspoint operation and line-control circuits are explained later in this chapter.

When a forward port is in the IDLE state, its nine I/O pads are enabled as inputs, and the data coming in to the forward port input flip-flops is latched on the falling clock edge [1].

[1] An explanation of the clocking of RN1 is given in Section 5.7

In the IDLE state, the crosspoint drivers are enabled so that data from the flip-flops is driven through the forward port's horizontal bus into the crosspoint array on phi2 rising. The data will not get any further unless a routing byte is recognized, in which case an `allocate` line is asserted, and the corresponding crosspoint is then armed to try to allocate a connection to a vertical back port bus.

Note: The main data path bus in the forward port, `netio`, is somewhat vestigial. Our original design had a single common bus going into the crosspoint array, and we continued this thinking to have a single internal bus in the forward port. The single bus datapaths in the crosspoint were changed to two separate busses, one for input and one for output. The forward-port state machine needs to examine data no matter which direction the port is in, backward or forward. Separating out the bus into a forward and backward bus would eliminate the time required to enable the tristate drivers from the input latches or the crosspoint buffers. This is a candidate for redesign in future versions of this chip.

5.2.1 Forward Port State Machine

The forward-port state machine has a mix of combinational and clocked logic. The clocked logic determines the next state, while the combinational logic can compute control outputs based on changing inputs during the course of a single clock cycle. Figure 5-3 shows a state-transition diagram for the forward port.

5.2.2 Early Allocate Datapath

According to our timing analyzer, the allocate cycle, in which a new connection is opened, was one of the critical timing paths on the chip. In order to make maximum use of the full clock cycle available, and to assure that certain combinational signals had settled before triggering dynamic logic in the crosspoint array path, we decided we needed to "get a peek early" at the new input data coming from the pads.

At the start of a clock cycle, as the input data arrives from the pads arrives during phi1, it is fed directly into a special "early" datapath. Normally pad data runs into a flip-flop and is simply latched internally during phi1, and not visible at the output until phi2. The early-out ports of the input flip-flops allow the allocate-early logic to see the data as it arrives during

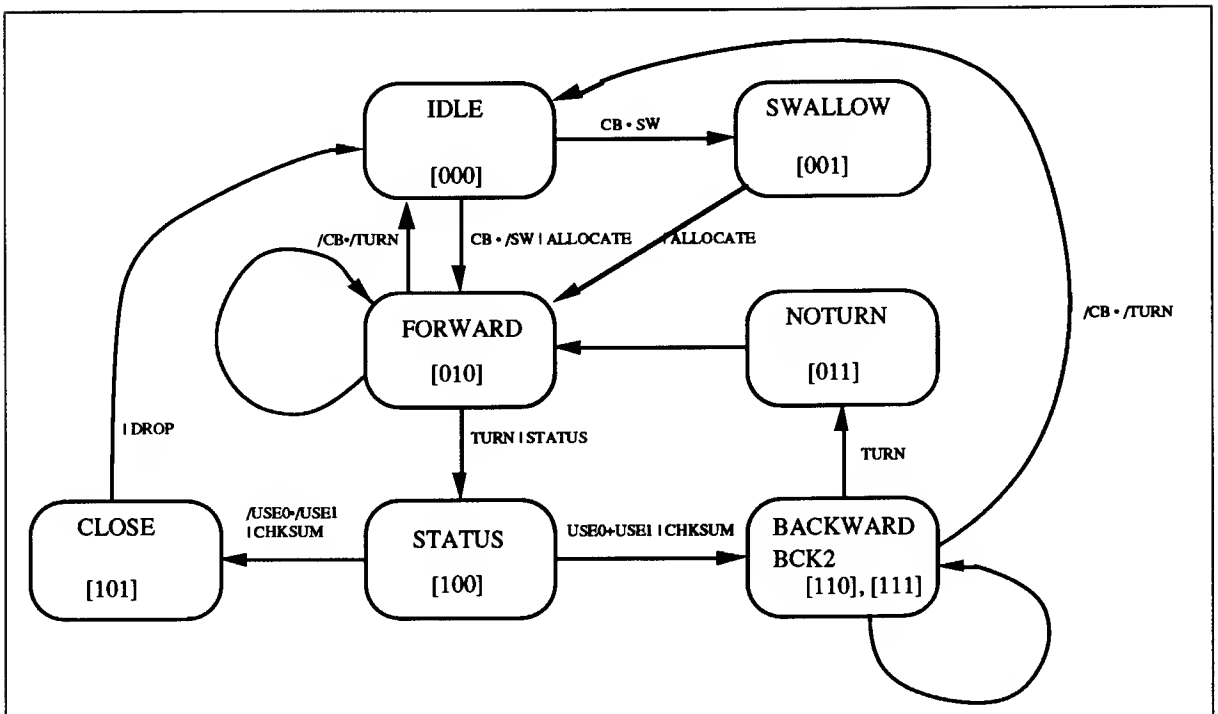


Figure 5-3: The forward port state-transition diagram.

phi1. The danger with looking at the data early is that the it is not yet stabilized by being latched, so it will have glitches during phi1. The `alloc` signal is generated by decoding the presence of a control bit, and the top two bits of the routing byte. As long as the setup time for phi1 is long enough for the incoming data to settle, the `alloc` lines will be stable when phi2 comes on.

The allocate-early module functions as a small independent state machine. Its job is to recognize a route command during phi1 (before the data has actually been latched into the input flip-flops), decode the routing destination, and enable one of four allocate control lines into the crosspoint array. Since the flip-flops in the allocate-early state-machine are processing data during phi1, it was neccessary to swap the clock inputs to its state flip flops, putting it a half cycle out of phase with most of the rest of the chip logic.

Figure 5-4 shows the basic circuit used for the allocate-early state machine. It consists of a flip flop and an AND and OR gate. In the IDLE state, a high control bit indicates a route command. This means an allocate operation must be initiated during the same clock cycle. The control bit, labeled as `early-cb`, comes directly from the pads. When the state flip-flop

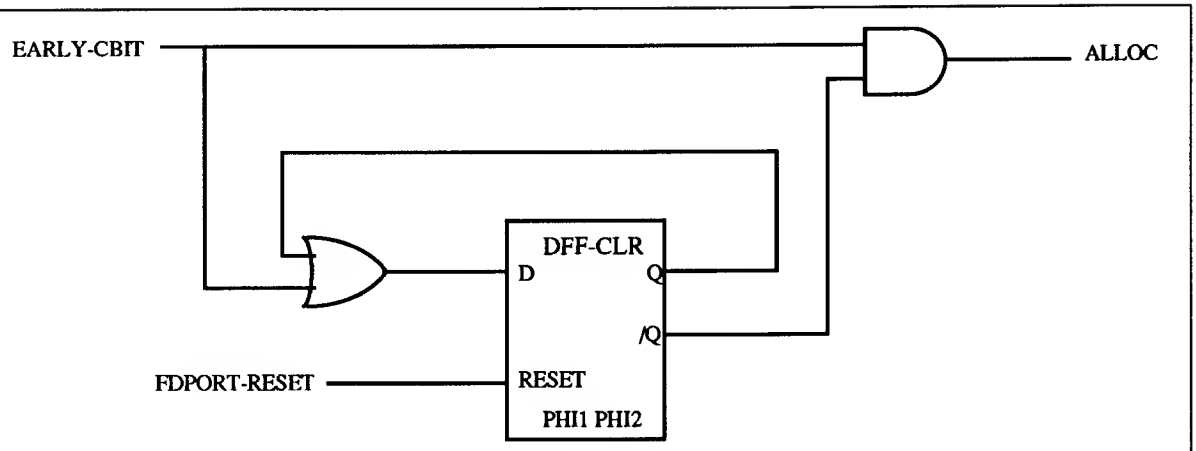


Figure 5-4: Basic allocate-early state machine.

is reset, the output enables the AND gate to pass an incoming control-bit. On the next cycle, the flip-flop will latch the high control bit value, and thus disables the AND gate from passing another alloc command. The OR gate passes the '1' around to be fed back into the flip-flop input, until the flip-flop is reset externally by the forward port state machine.

The presence of an extra state in the allocate cycle when the *swallow* state is enabled creates the need for a little more complexity in the allocate-early hardware. Figure 5-5 shows how an extra flip-flop was added to the circuit to enable a one cycle delay in the allocate sequence. The first flip-flop enables the second, and the second flip-flop actually asserts the alloc signal. A pair of muxes select between the one and two cycle allocate sequences, depending on the state of the swallow control signal.

In order to set up for the next routing byte, the allocate-early state machine must be reset when the connection is dropped, or a blocked connection is turned. Tragically, our test vectors did not cover this last rather simple case in the RN1 chip, and thus the allocate-early flip-flop is not reset under the previously mentioned circumstance. See the state diagram in Figure 5-3 for the guilty state-transition.

5.2.3 Checksum

The checksum unit computes a 17 bit internal checksum, and reads out the bottom 14 bits in the status and checksum bytes during a turn. The checksum unit has seventeen flip-flops linked in a shift-register, with XOR feedback taps at bits 16 and 4. The bottom eight bits also have

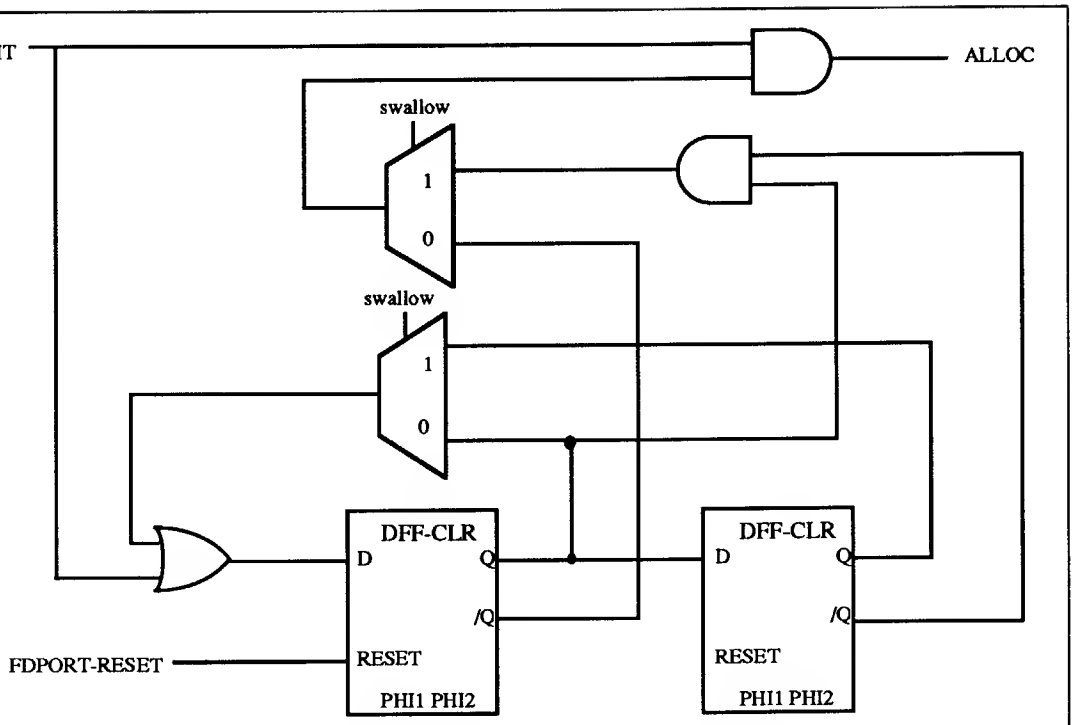


Figure 5-5: Basic allocate-early state machine.

the sequential data words XOR'ed in each cycle with the feedback taps.

Appendix A details the checksum algorithm as a lisp function. The checksum function was designed to emulate a maximal length sequence pseudorandom number generator. Note that the clock is always running on the checksum circuit, so that during a turn, between the first (status) and second (chksum) byte of the checksum, the shift-register is clocked once. When designing hardware to verify the returned checksum, the reader may test his design against the values shown in the test-vectors in Appendix D. Note that the next revision of the chip, RN1b, will use the CRC-CCITT polynomial checksum algorithm.

5.3 Back Port Datapath

The back port is quite similar to the forward port. It has access to its pads, and data busses running into the crosspoint array. Figure 5-6 shows a block diagram of the datapath.

There are a few circuits in the back port that require explanation. The use line is a status line which comes up out of the crosspoint array. It indicates that this output column was

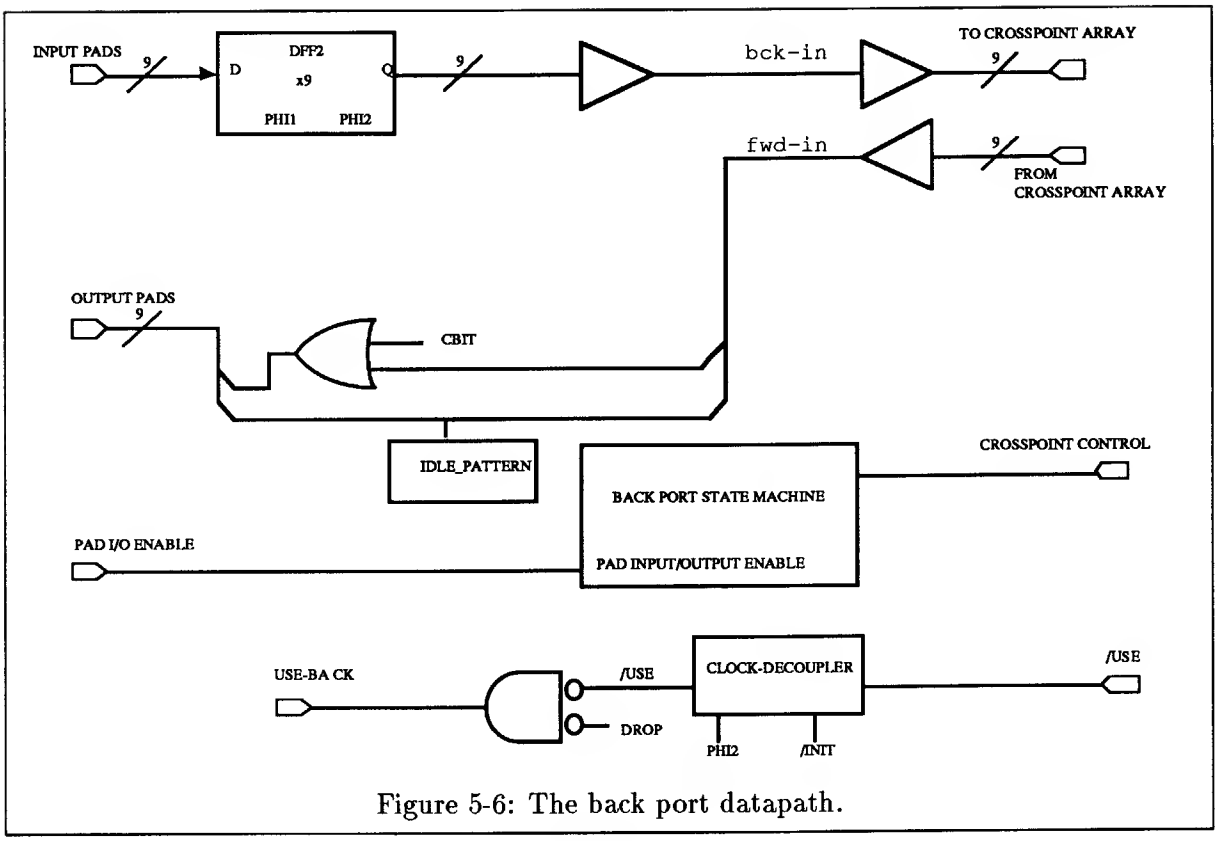


Figure 5-6: The back port datapath.

grabbed by one of the eight forward ports, and that active data will be appearing during this cycle. Since the use signal is a dynamic precharged line, it is important that the back port only examine the value during the evaluate cycle (phi2). During the precharge (phi1) the value will of course be logic high. The `clock-decoupler` circuit is a latch which allows data to flow through during phi2, and holds the value during phi2 low.

The use signal is sent back down to the line control unit at the base of the output column. This allows the line control to decide whether to make the output column available for routing by other forward ports during the next cycle.

Unlike the forward port, there are two fully independent nine-bit busses in the back port for data coming off the back-port pads, `bck-in<8:0>`, and data coming from the crosspoint, `fwd-in<8:0>`. As noted earlier, this is probably the right model for the forward port as well. Perhaps someone will take this to heart in the next design of the switch chip.

5.3.1 Back Port State Machine

The back port state machine, `out-fsm-std`, is slightly simpler than the forward port state machine. It mainly has the task of watching for use, turn, or drop signals on the `fwd-in` or `bck-in` busses. The state transition diagram is shown in Figure 5-7.

As in the forward port, there is a special circuit to detect a turn byte. Figure 5-8 shows the turn and drop detector circuits in the state-machine module. Since the forward and backward data busses are completely separate, there are two turn units. The back port state-machine also needs to be able to detect a drop quickly. This is a case where the control bit is low and the data bits are not all high. To minimize glitches we needed the turn unit to produce its result simultaneously with a unit to detect the dropped control bit. Since we were not strapped for space, we duplicated the turn detect circuits, and just fed in the control bit and its inversion into the duplicate units. This produces the `fwd-turn` and `fwd-cb`, and `bckturn` and `bck-cb` signals almost simultaneously.

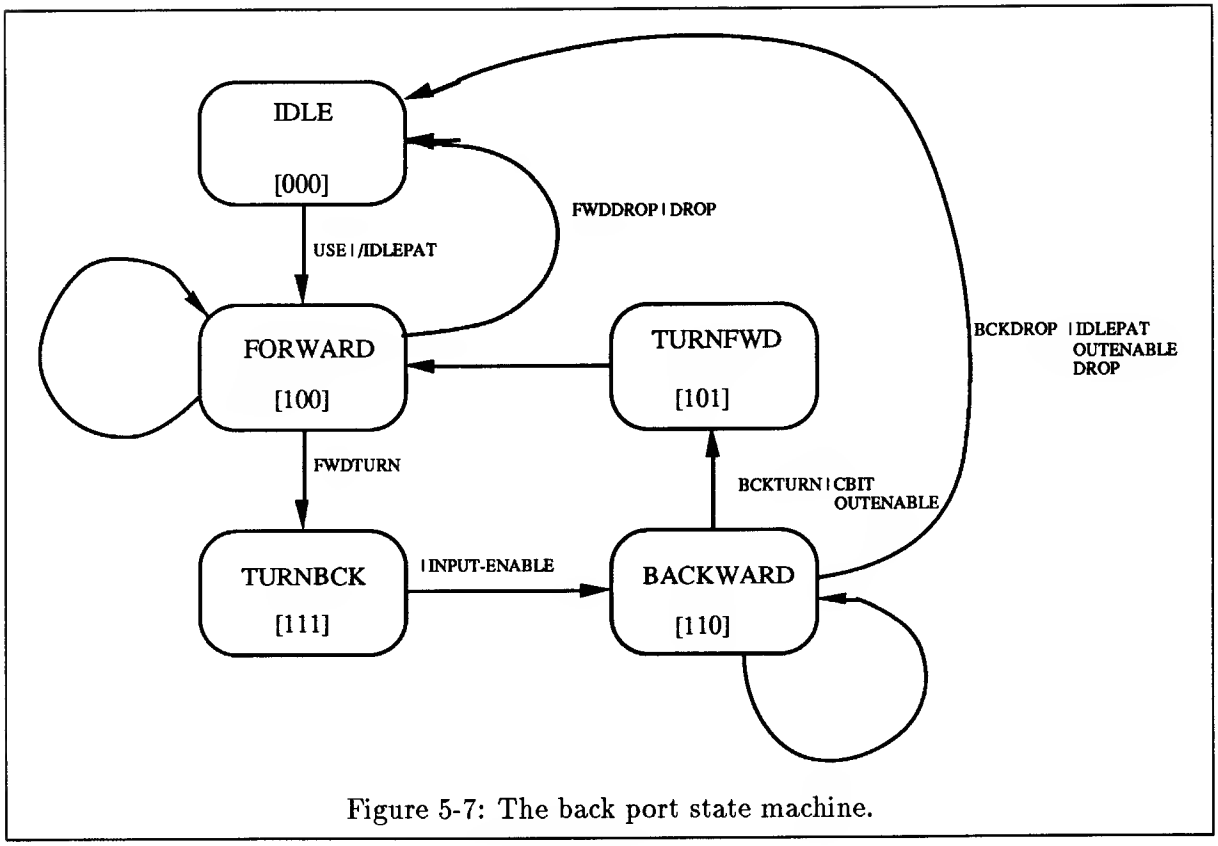


Figure 5-7: The back port state machine.

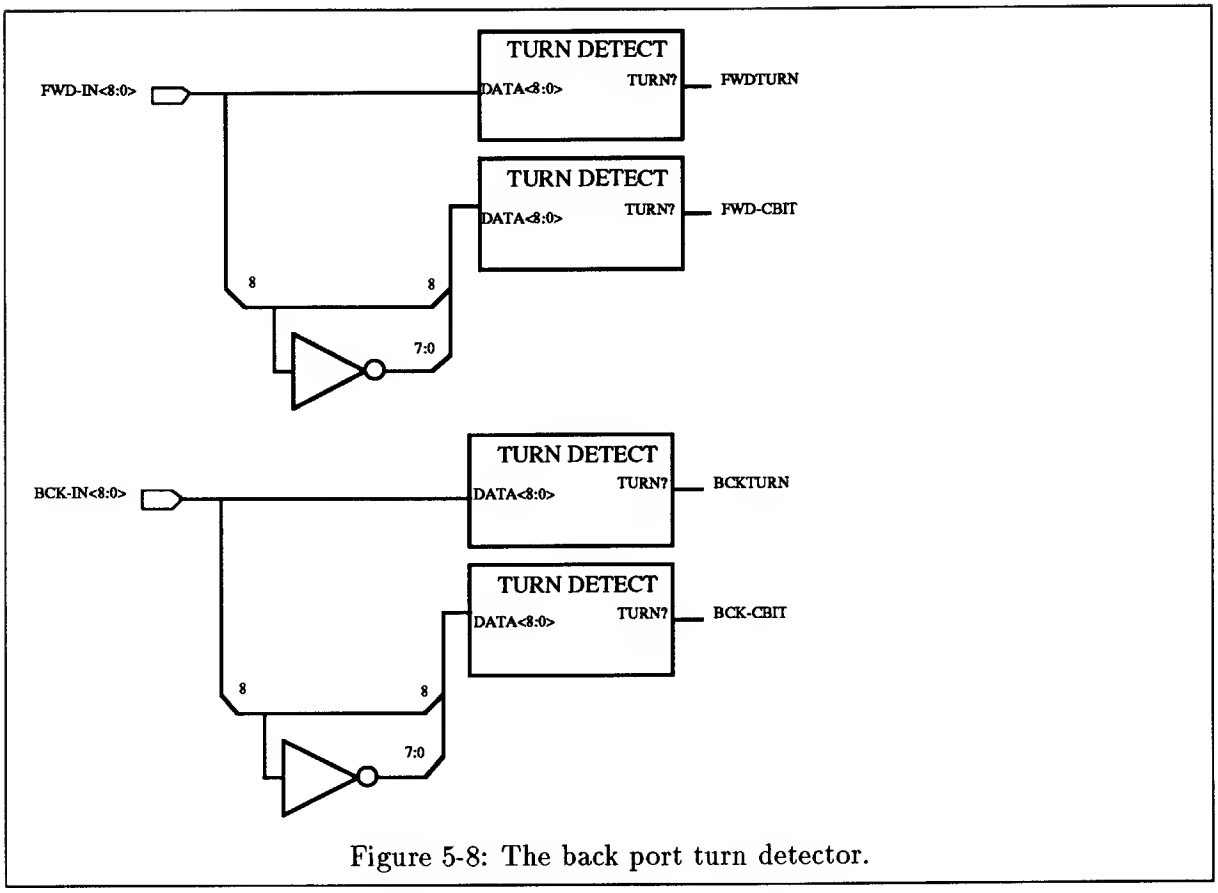


Figure 5-8: The back port turn detector.

5.4 Crosspoint Array

The crosspoint array is the heart of the RN1 switching chip. It provides the connection fabric through which any forward port may connect to any back port. It is organized as an 8 x 4 matrix of crosspoint modules. Figure 5-9 shows the layout of the array. The arrows indicate the direction of control signal flow. Horizontal arrows show the allocate lines which are generated from the forward ports. The vertical arrows show the direction of propagation of the `line-ctrl` control signals. In order to make best use of the pads available on all four sides of the chip, the forward ports alternate in location along the left and right sides of the chip, and the back ports alternate on the top and bottom sides of the chip.

5.4.1 Crosspoints

Figure 5-10 shows a block diagram of a crosspoint. Each crosspoint serves one forward port, and has access to two back-ports. The module consists of two sets of tristate nine-bit bus drivers, and the allocate logic. The tristates are what actually connects a forward-port to a back port. They are under control of the allocate logic. The use-line modules are used to quickly inform the back-port if it has been acquired by a crosspoint, by pulling the `using0` or the `using1` line low.

Allocate Logic And The Line-Control Modules

In order to understand the allocate logic in the crosspoints, it is necessary to understand the information that is fed to them from the line-control units at the base of each crosspoint-array column. The crosspoint array is arranged as 8 rows of 4 columns. Each line-control unit at the base of a column serves a pair of back-ports (Figure 5-11) and eight crosspoints. Thus there are four line-control modules in all. Each column is said to have two channels, with each channel connecting to a back-port. The back ports are grouped in pairs corresponding to logical routing directions. Table 5.1 shows the grouping of back ports to logical routing directions.

Each cycle, the line-control module for a pair of back ports looks at which, if any, of its two back-ports are free from the last cycle. It then generates four control signals which propagate up the column telling the crosspoints which are the allowed back-ports to choose, should they

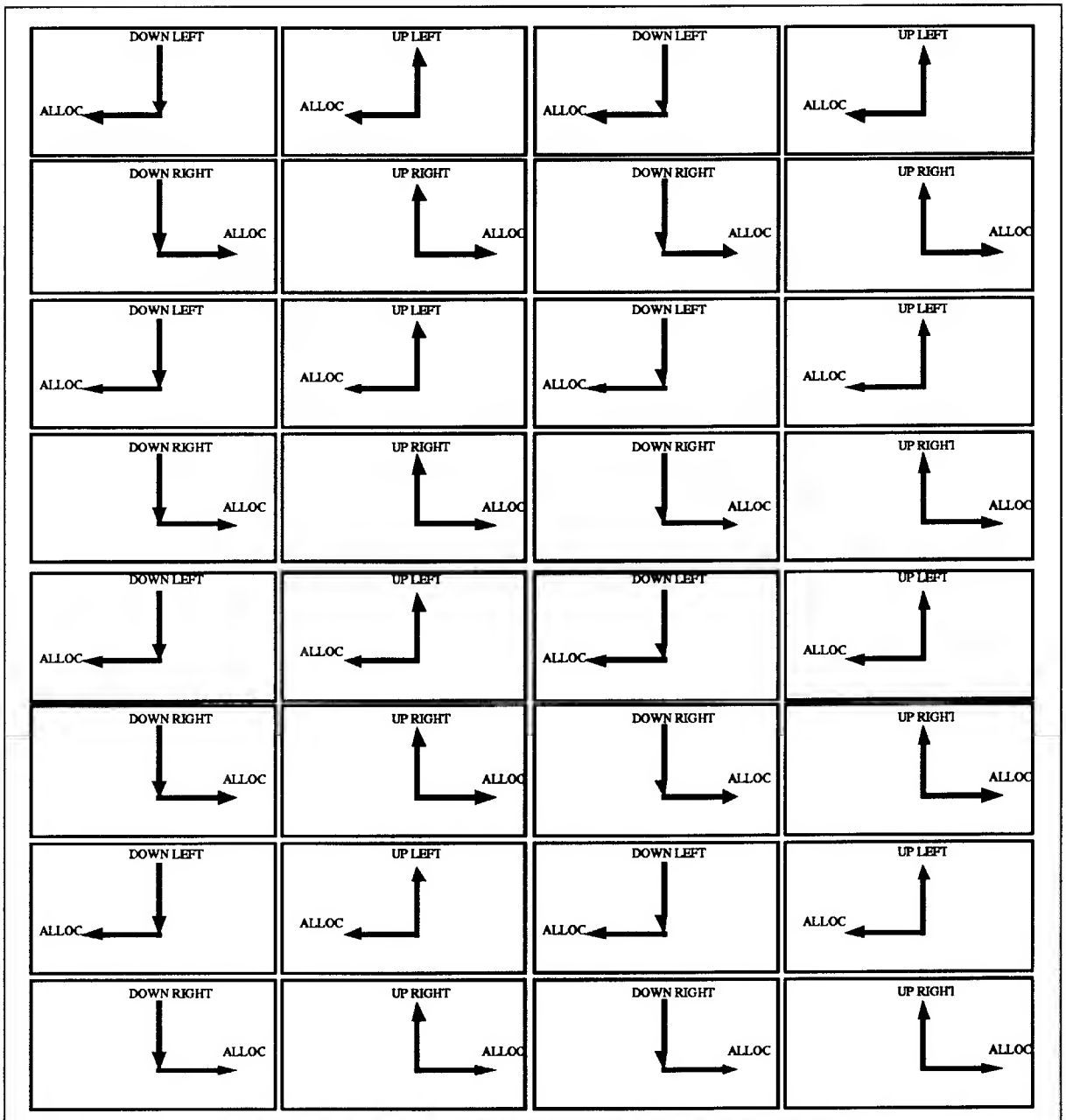


Figure 5-9: The central crosspoint switching array.

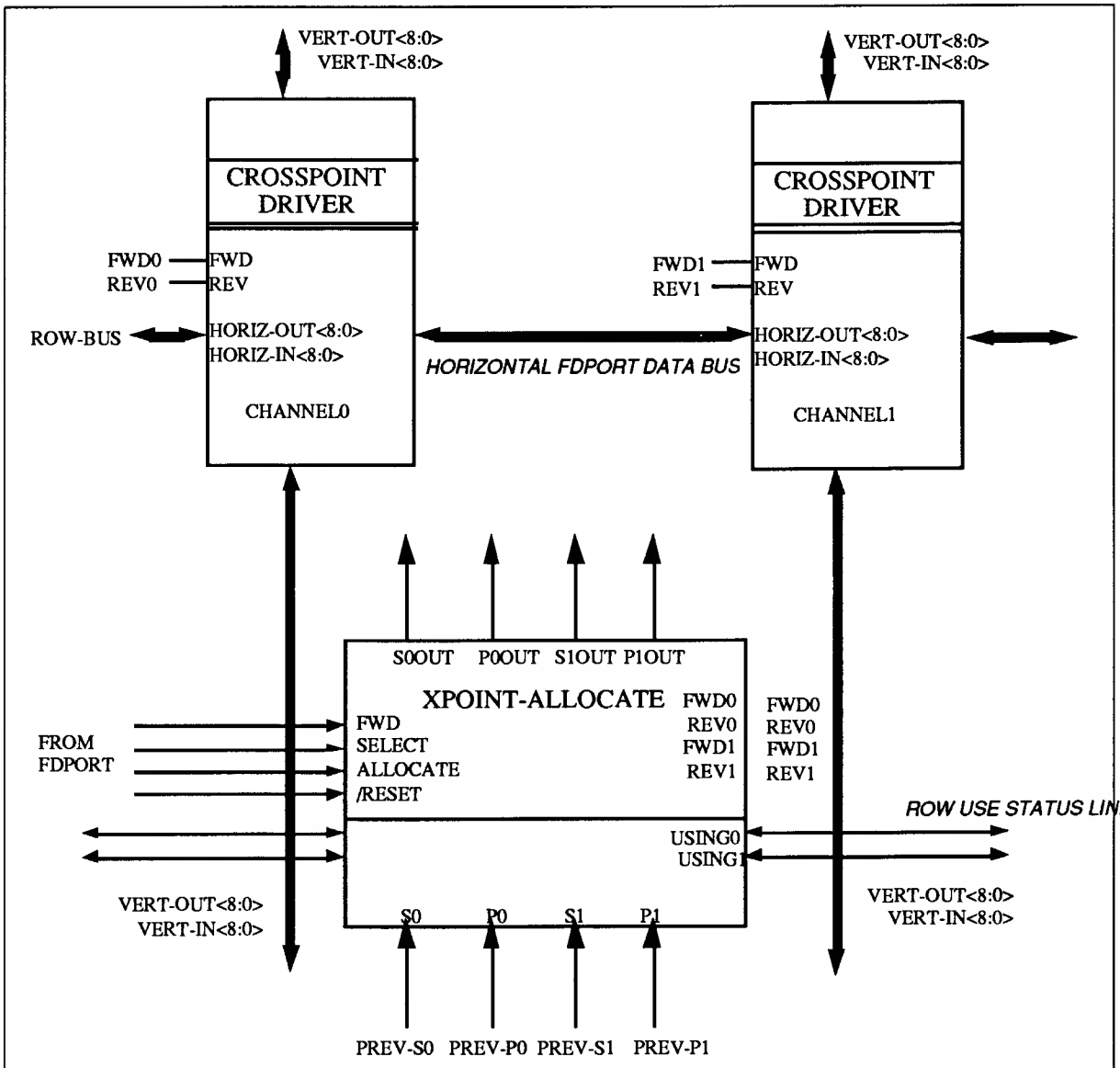


Figure 5-10: A crosspoint module.

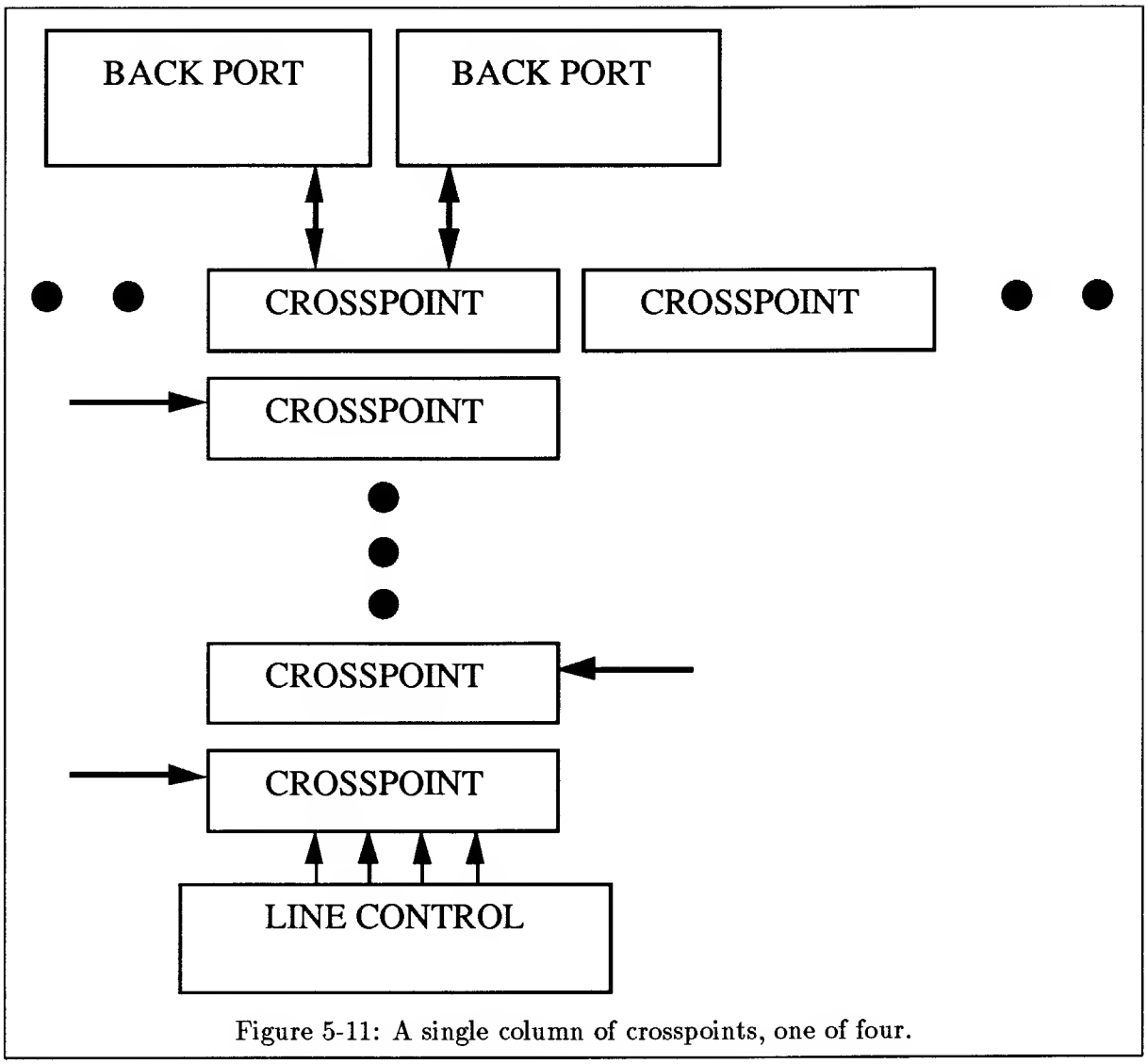


Figure 5-11: A single column of crosspoints, one of four.

Back Ports	Logical Routing Direction
[0,1]	0
[2,3]	1
[4,5]	2
[6,7]	3

Table 5.1: Back Ports Logical Routing Direction

wish to try and open a connection.

A line-control module is shown in Figure 5-12. The module is responsible for telling the crosspoints in its column which back ports are free, and if both are free, which is the preferred one to choose first. It does this by generating the four control signals`p0`,`p1`,`s0`, and `s1`.

5.4.2 P And S Control Signals

The meaning of the `P` and `S` control signals for 8x4 routing mode is shown in Table 5.2. The letters 'P' and 'S' stand for 'primary' and 'secondary'. In the case where only one channel is free, it will be selected as the primary channel. When both channels are free, a choice is made as to which will be the primary channel, based on the internal pseudorandom number generator.

p0	s0	p1	s1	Meaning
0	0	0	0	no channels free
1	0	0	0	channel 0 free
0	0	1	0	channel 1 free
1	0	0	1	both channels free; 0 preferred
0	1	1	0	both channels free; 1 preferred
X	X	X	X	no other legal encodings

Table 5.2: P0, S0, P1, S1 Encodings For 8x4 mode (SELECT = 1)

This is to help improve routing statistics and fault tolerance in the Transit network. If a message is blocked by a busy port or faulty chip, there is an approximately even chance that the next attempt to open the connection through this chip will choose the other logically equivalent channel.

As the P and S signals propagate up the column, a crosspoint can grab only a primary channel. When a primary channel is grabbed, the associated secondary channel, if any, is then 'promoted' to be a primary channel, by shunting the signal on its S line to its P line. The details of the logic to implement this allocate circuit are explained in the next section.

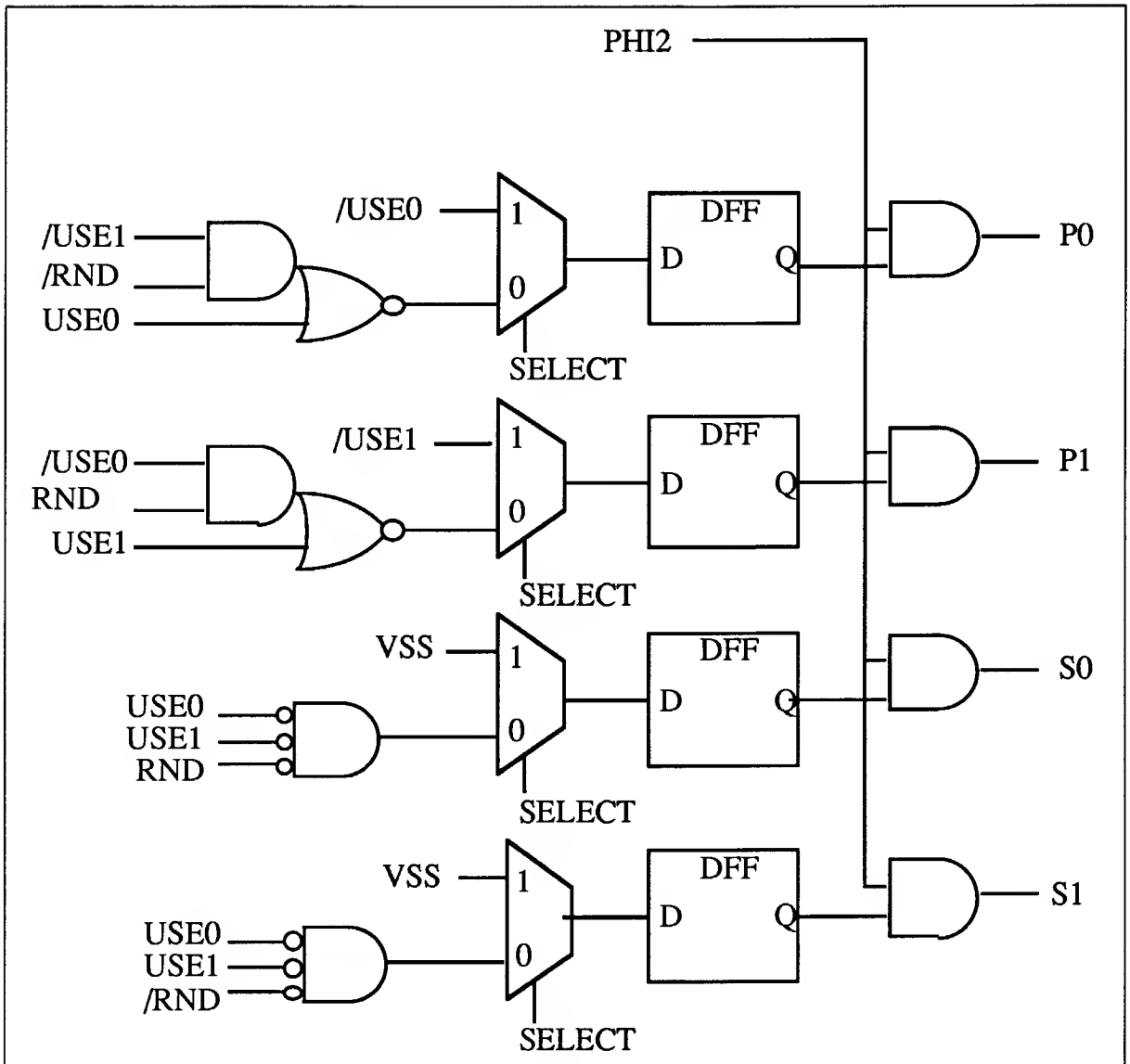


Figure 5-12: A line-control module.

5.4.3 Independent 4x4 Routing Mode

The logic for allocating free channels becomes trivial for the independent 4x4 mode. The select control signal is low, causing the line-control muxes to generate `p0` and directly from use0 and use1. There are no secondary channels in this mode, so the `s0` and `s1` lines are always deasserted.

Table 5.3 shows the valid encodings for the line-control signals in dual independent 4x4 crossbar mode. For each crosspoint row, in dual 4x4 mode, all the crosspoints belonging to even rows (forward ports 0,2,4,6) will look at and allocate from only the even back ports (`p0` lines). The crosspoints belonging to the odd forward ports will allocate from the `p1` lines.

p0	s0	p1	s1	Meaning
0	0	0	0	no channels free
1	0	0	0	channel 0 free
0	0	1	0	channel 1 free
X	X	X	X	no other legal encodings

Table 5.3: P0, S0, P1, S1 Encodings For 4x4 mode (SELECT = 0)

The allocate logic in one of the four crosspoints in a row is activated when the row forward port wants to open a connection to a back port. The grant logic in a column is shown in Figure 5-13. Given the status information on the P0, S0, P1, S1 lines, the allocate hardware of each activated crosspoint trys to gain exclusive control of a channel. The P and S signals propagate up through the eight successive allocate modules in a column. The grant logic is inherently unfair, with allocate modules closest to the line-control unit having first crack at grabbing the channels.

To understand how an allocate cycle works, let's look at single column during the course of a clock cycle. Assume that forward ports 3, 4 , 5, and 7 are all trying to open connections in this column during this cycle. Assuming both channels are free from the last clock cycle, at most two of these four contenders can successfully allocate channels for their exclusive use. Figure 5-14 shows the progression of the P and S status lines up the column during the course of the cycle.

The two crosspoints closest to the line-control were the successful winners of the free channels. As was stated before, the grant logic is inherently biased towards crosspoints closest to

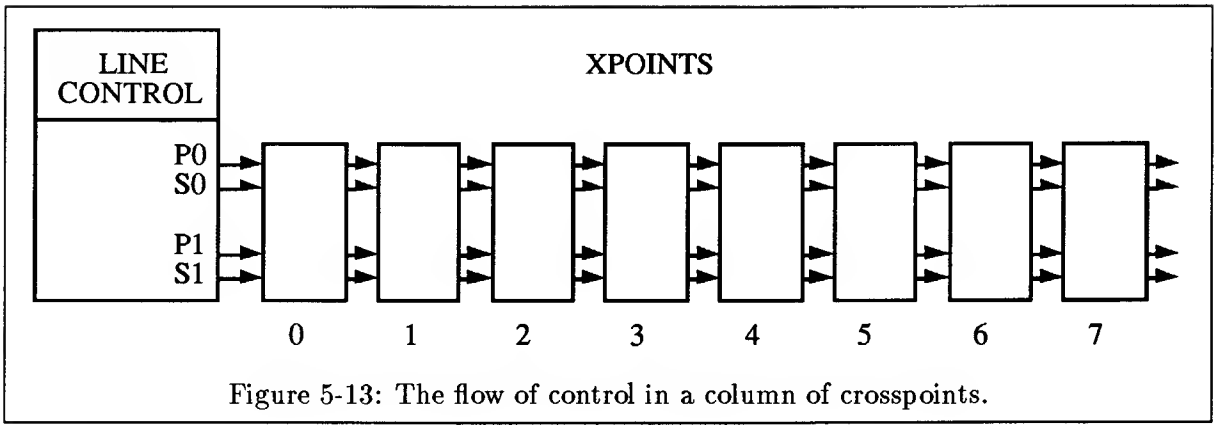


Figure 5-13: The flow of control in a column of crosspoints.

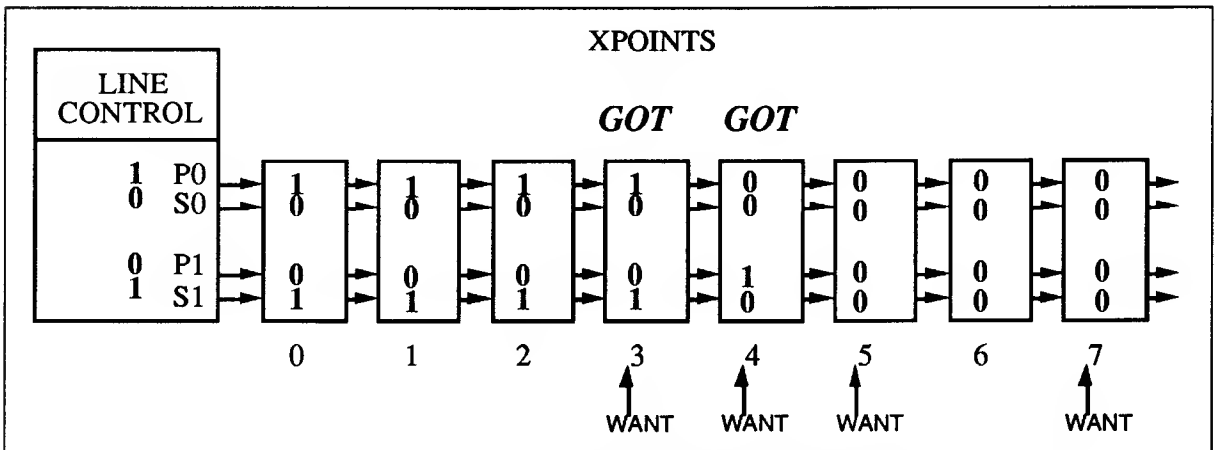


Figure 5-14: The flow of grant control signals up a column of crosspoints.

the base of the column. The logic was designed this way to get maximum speed through simplicity. The unfairness is mitigated somewhat by the folded topology of the crosspoint array. Each forward port has a row of four crosspoints. A forward port which has two crosspoints x steps away from line-controls will have the other two crosspoints $(8 - x)$ steps away from the other two units. Thus there are four distinct classes of forward ports on each chip, as shown by Table 5.4.

Class	Crosspoint Priorities
1	1, 8
2	2, 7
3	3, 6
4	4, 5

Table 5.4: Classes of crosspoint grant priorities.

The effects of this unfair allocation can be further mitigated by wiring the duplicate back ports in each column to different classes of forward ports in the next stage of the Transit network.

5.4.4 Allocate Logic In A Crosspoint

Figure 5-15 shows half of the allocate logic in a crosspoint. The circuit is duplicated for the two channels in each column. The carry-units are the heart of the allocate grant chain.

The control signal allocate is aliased internally as the `want` signal. The `want` signal is fed to one or both carry-units, depending on the `select` signal. Recall that `select` chooses between 8x4 dilation 2 mode and dual independent 4x4 router mode.

When `select` is asserted high, meaning dilation 2 mode, the `want` signal is passed to both carry units. The `want` signal means what it says; that this module wants to grab control of a back-port bus. The allocate logic arms both of its carry units with the want signal, and waits for the `p0,p1` signals. After the dust has settled in an allocate cycle, the "got" logic determines if the module actually got control of a bus. This information is immediately used by the crosspoint's bus drivers to enable data onto the bus.

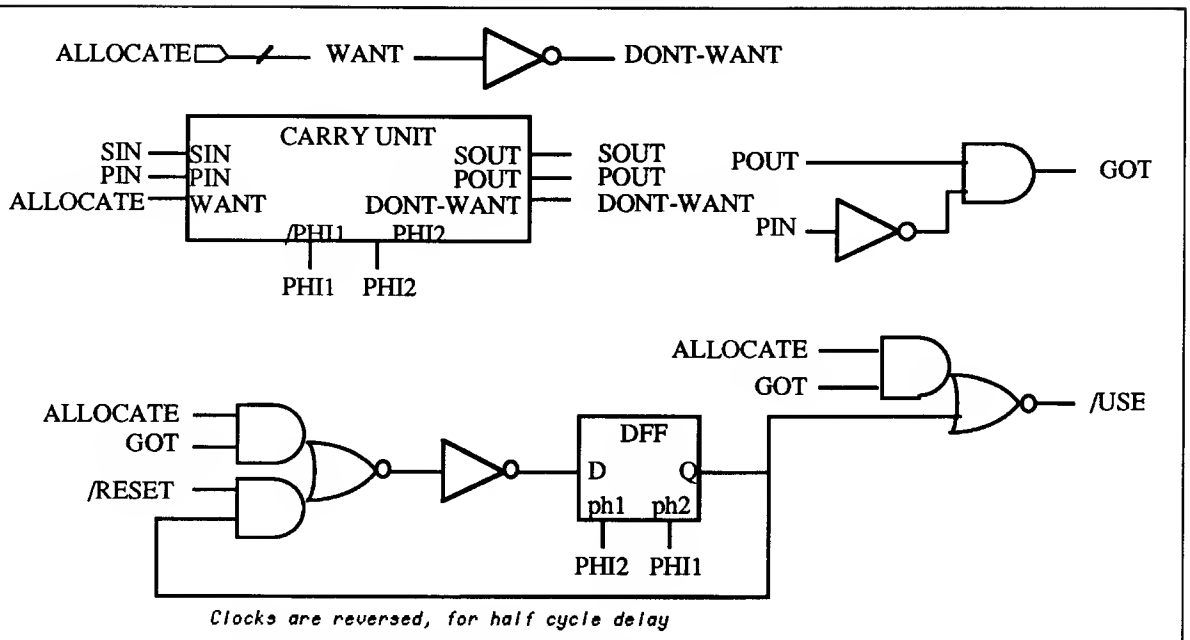


Figure 5-15: The allocate logic in a crosspoint.

5.4.5 Dual vs. Independent Allocation

The global control signal `select` determines whether the chip is in 8x4 (dilation 2) mode or dual independent 4x4 mode. The action of the `select` signal is very simple. It does exactly two things to the chip. First, it determines the generation of the `P` and `S` control signals in the line control module. Second, it acts in each crosspoint to determine if one or both of the allocate units are active.

When `select` is low, meaning independent 4x4 router mode, the two carry units in each crosspoint are decoupled, and only one is ever activated. Crosspoints in *even* numbered rows will only arm their channel A carry unit, and crosspoints in *odd* numbered rows will arm their channel B carry unit. Thus, even numbered forward ports will only be able to allocate even back ports, and odd numbered forward ports can only allocate odd back ports. Figure 5-16 shows the logic for the select mode programming.

When `select` is high, meaning 8x4 (dilation 2) mode, the two carry units in each crosspoint are both armed. The line-control unit makes sure that the `p0, p1` signals never indicate that both back ports in a column are primary allocate choices.

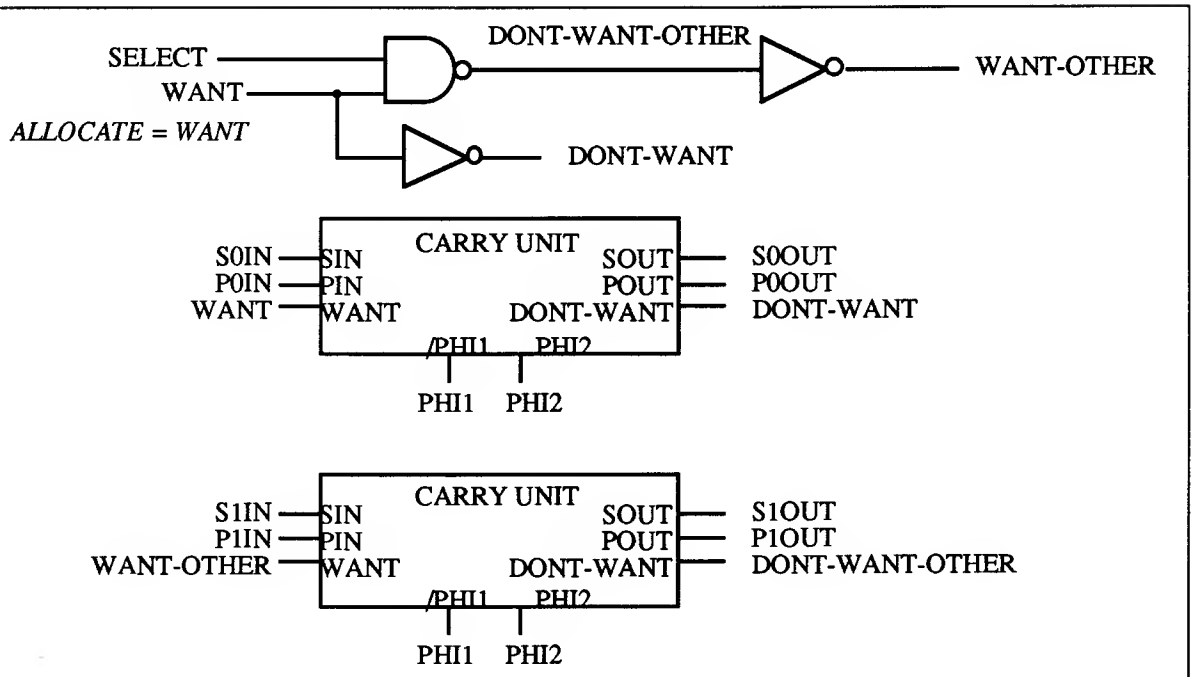


Figure 5-16: The `select` logic decides if a crosspoint is in 8x4 or 4x4 mode, and the parity of a crosspoint hardwired into each crosspoint to make it an *odd* or *even* crosspoint.

5.4.6 Precharged Bus Lines Enhanced With Positive Feedback

The actual implementation of the allocate grant logic was done using dynamic logic, for speed. Originally the allocate circuits were done in static combinational logic. The associated gate delay, multiplied by eight crosspoints in a column, proved to be a target for optimization.

In the dynamic logic design, the P0,S0,P1,S1 lines are precharged busses, and the active grant signals are propagated as low going pulses. Figure 5-17 illustrates the basic grant propagate sequence during a clock cycle.

The `SIN` and `PIN` lines are connected to the `SOUT` and `POUT` lines of the module below or, in the case of the bottom crosspoint, to the line-control unit. The `S` and `P` lines are precharged during PHI1. On PHI2, low going pulses are sent of the line-control unit for active `P` or `S` signals.

If this crosspoint does not want to allocate, the `DONT-WANT` signal is asserted, allowing the pulses to pass through untouched. The positive feedback inverters sense a low going pulse, and help to slam the line down to ground, thus boosting the pulses as they travels up the column.

In the case where the crosspoint does want to allocate, the actions are a little more subtle.

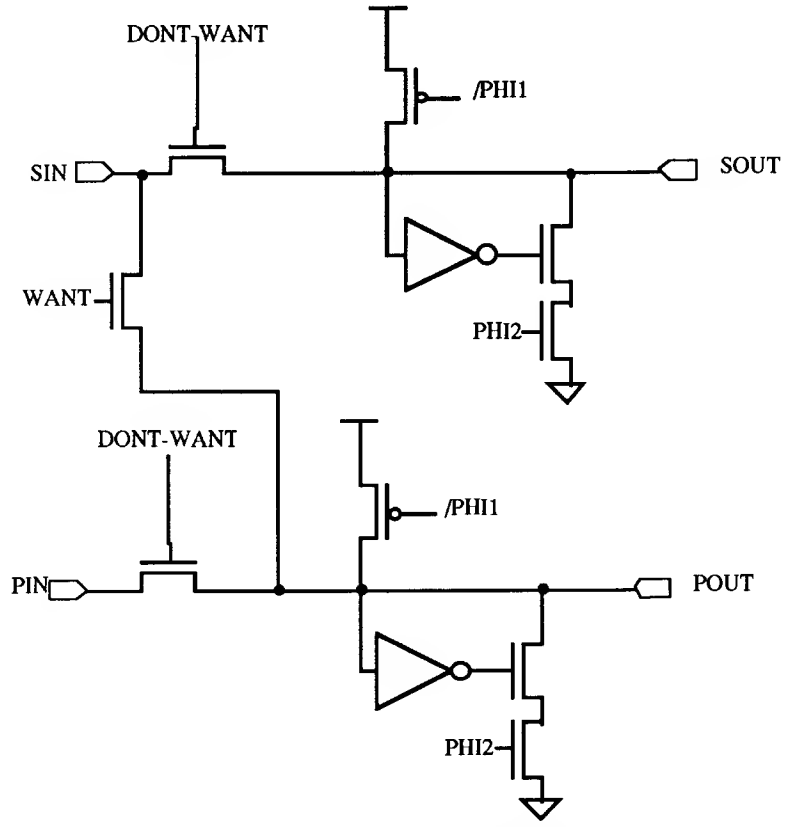


Figure 5-17: The basic dynamic logic circuit in the allocate carry grant chain.

DONT-WANT is held low, and WANT-EITHER is asserted. There are three possible cases which can happen at the carry unit, depending on the P and S lines.

1. No pulses are coming in either the P or S lines This channel is not free. P and S stay charged up. Nothing happens.

2. A pulse is coming down the P line. This channel is free, and it is the primary choice for allocating. The P pulse is simply stopped dead in its tracks. This is how a crosspoint gains control of the bus. If the pulse reached here, it means no previous crosspoint got control. Stopping the pulse means no downstream crosspoint will see it.

   The GOT logic in a crosspoint can look at the PIN and POUT line, and notice that PIN is low and POUT is still high. This implies that the buck has stopped here; this crosspoint has acquired control of this bus.

3. A pulse is coming down the S line. This channel is free, but it is the secondary channel. The carry-unit logic stops the S pulse, and shunts it over to the P line. This guarantees that any downstream crosspoints will see this channel as available.

Note that by Table 5.2 , if S is asserted in one channel of a column, the other channel of the column pair will have its P signal asserted. This is because if you have a secondary channel, it implies that the other channel is the primary channel.

5.4.7 USE Line Logic

The two use lines in a column indicate whether any crosspoint has acquired the A or B channel. These lines are logically eight input OR gates. The eight stages in the column make this a slow gate to implement in CMOS. For this reason, I implemented the use logic using a precharged bus in a manner similar to the allocate logic. Figure 5-18 shows the logic in half of a crosspoint stage. The circuit precharges the one bit bus during phi1, and conditionally discharges it during phi2, based on the got signal from the allocate logic. The circuit uses positive feedback to help to slam the bus low when it detects it dropping. This creates a self-propagating low going pulse down the the line.

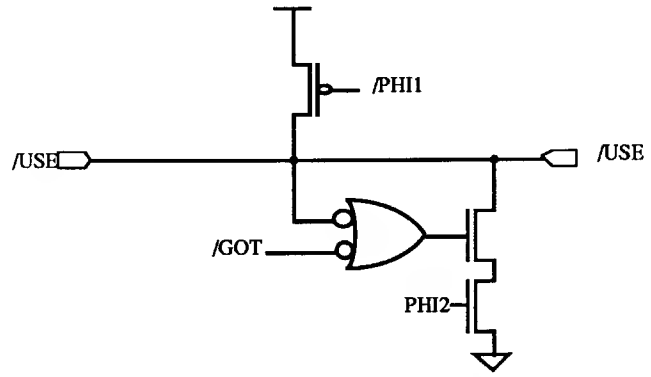


Figure 5-18: The dynamic wired OR structure of the crosspoint column USE signal.

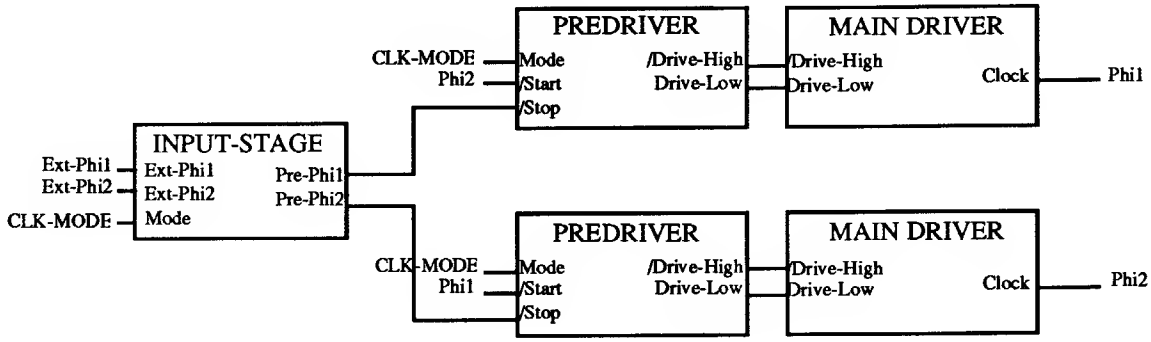


Figure 5-19: The clock generator can be programmed to create a two phase non-overlapping clock off of the external PHI2 signal, or can supply the chip with the external PHI1 and PHI2 signals.

5.5 Clock Generator

The RN1 chip uses internally a two phase nonoverlapping clock. The clock signal is derived from an external clock supplied to the clock pads. The clock generator has two modes, selectable through the clock-mode input pad to the chip. With clock-mode low, the two phases of the clock must be supplied independently on the ext-phi1 and ext-phi2 pads. As it happens, the signals must be supplied inverted. This gives the system designer maximum control over the non-overlap period of the clocks. The clock generator circuit is shown in Figure 5-19.

The three modules, `INPUT-STAGE`, `PREDRIVER`, and `MAIN-DRIVER` are shown below.

When `clock-mode`is set high, the mux ignores phi1 completely, and derives all timing off of the ext-phi2 signal.

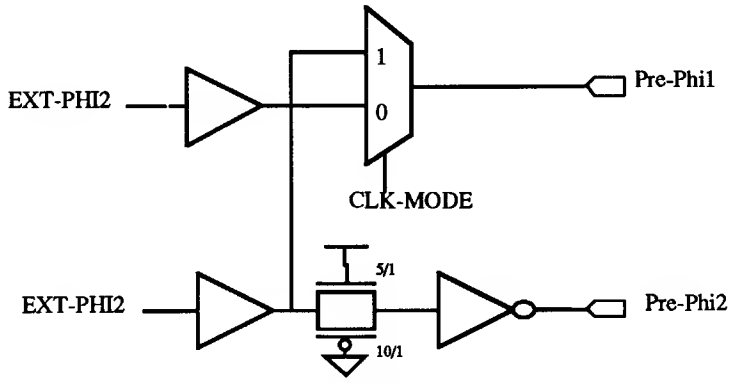


Figure 5-20: The input stage of the clock driver.

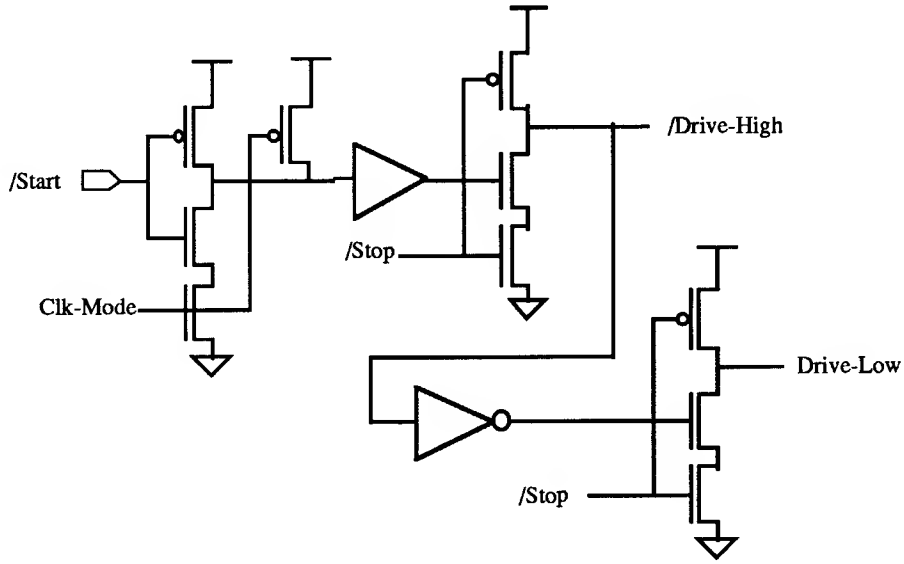


Figure 5-21: The clock line predriver circuit.

5.6 Pad Drivers

The pads used in RN1 are 5 volt CMOS bidirectional drivers, originally desgined by Hewlett-Packard, and modified by Symbolics. The 1V CMOS pad design in being tested on a separate chip, and will be incorporated into the next CMOS Transit routing chip.

On the pads as we got them from Symbolics, there are numerous enables and latch features which we did not need to make use of. The nine data lines in each of the eight forward and back ports are all configured as simple bidirectional drivers. One pair of test pads was placed in the pad ring. These are the test-in and test-out pads, a simple loopback, where data into input pad test-in is routed around and output test-out. This provides a measure of on-off chip delays.

Simulation in SPICE gives a 4-5ns on-chip and off chip delay for the pads. This effectively limits the cycle time of the entire system to significantly less than 100 MHz.

5.7 Clock Timing And Data Transfer

RN1 has an on chip clock generator to create its internal two-phase clocks. For chip testing, the clock generator can be bypassed, and a two-phase non-overlapping clock can be applied externally.

When the on-chip clock generator is enabled (CLK-MODE pin tied to VDD), the system clock is applied to the PH2 pin, and RN1 generates its internal two-phase clock from this signal. The timing relations of the internal clock to the externally supplied clock is shown in Figure 5-22. The clock-generator circuits are detailed in Section 5.5.

When the clock generator is disabled (CLK-MODE tied to VSS), the clock timing is as shown in Figure 5-23. PH1 and PH2 are active low, and must be non-overlapping as shown.

Most flip-flops in the RN1 chip are simple D-type flip-flops, as shown in Figure 5-24. The flip-flops are clocked on a two-phase non-overlapping clock. By convention, phi1 latches the data internally into the first stage, and phi2 enables the new data at the output.

With the internal clock generator enabled, the flip-flops connected to the pads latch their data a fixed time delay after the fall of external clock. Figure 5-25 shows the timing for a sequence of input data bytes to a pad.

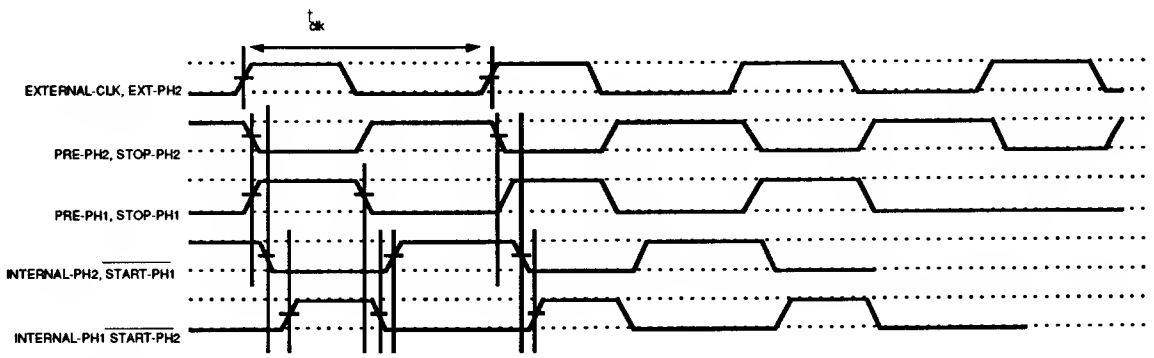


Figure 5-22: Relation between external clock and internal clocks with on-chip clock-generator enabled.

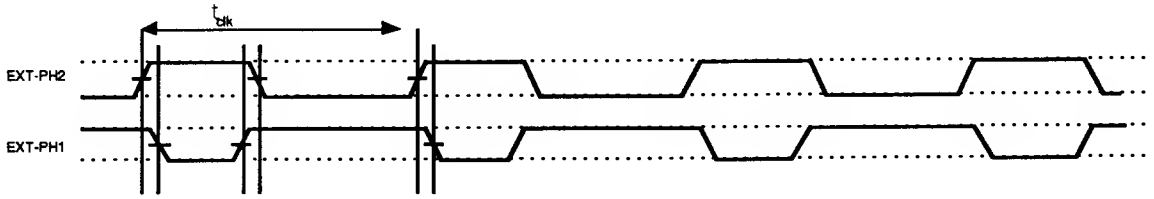


Figure 5-23: Externally supplied two-phase clock waveforms.

The data pads of each forward and back port are bidirectional. A turn command causes them to change from inputs to outputs. Figure 5-26 shows the timing of a turn.

The output pads at the far side of a connection can also change from outputs to inputs. For example, when a backward connection is dropped or turned, the forward ports change from outputs to inputs. Figure 5-27 shows this sequence of data bytes.

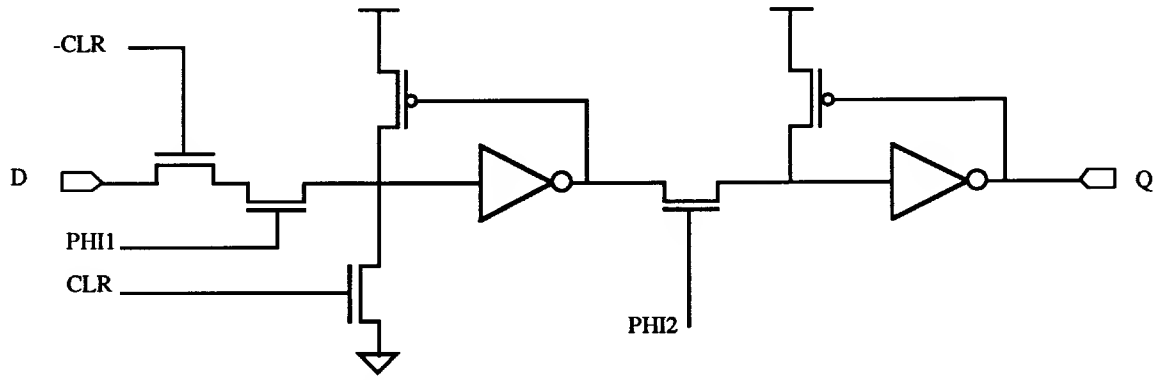


Figure 5-24: The basic flip flop, using a two phase clock.

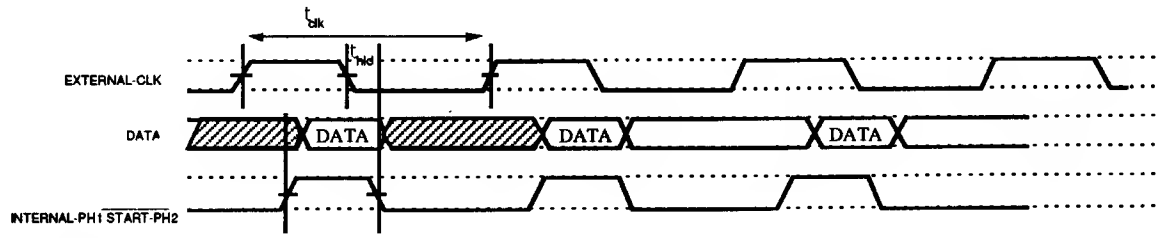


Figure 5-25: Data being transferred from an external driver to a port in input mode.

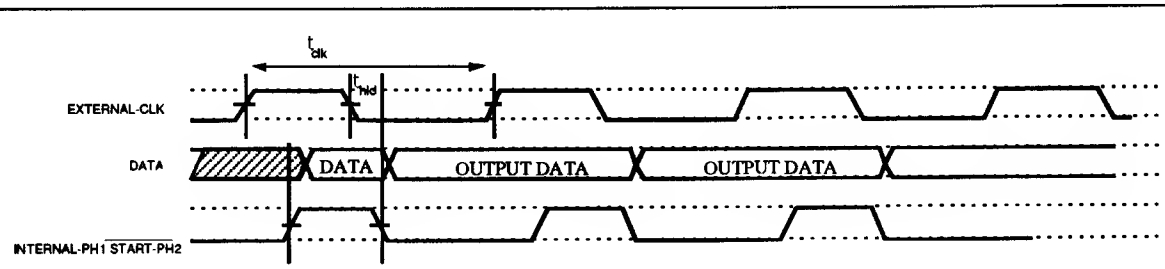


Figure 5-26: Data transfer during a turn

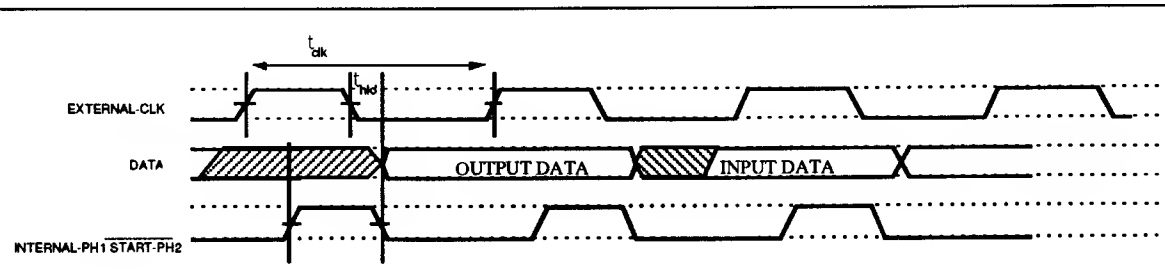


Figure 5-27: Data transfer during a backward turn

Chapter 6

High Performance Circuits: Design And Testing

One of the purposes of implementing RN1 in custom CMOS was to gain performance over what was possible from commercially available gate-arrays. Aside from the high pin count of our chip, the logical function would most likely fit in a large (50-100k gate) gate array.

Using custom CMOS allowed for tuning of circuit performance, as well as one-of-a-kind circuits to perform logical functions faster than library gates. As a first pass, we designed the logic using library standard cells, with fully complementary static gates. Using the NS timing analyzer, PEARL, we identified critical paths in the allocate logic and decided to optimize the speed using several techniques.

- Buffering large capacitive loads.

  In cases where a node had a large fanout, or a long distance route, we added one to three stages of exponential buffering. We used SPICE to analyze speed improvements.

- Asymmetric Transistor Sizing.

  In several cases, the critical path involved the delay for a gate to be pulled in just one direction. In these cases, copies of the standard-cell gate were made, with only the PMOS or NMOS transistors sized to pull in the required direction. This reduced the capacitive load on the drivers while increasing the drive in the right direction.

- Dynamic Precharge Logic.

  In cases where a signal was required to propagate through eight stages of static logic, we found that we could achieve a four to eight fold speed increase by implementing the logic as precharged buses with positive feedback. The allocate grant chain and use line logic were implemented in this way. There was increased complexity in interfacing to the static logic; going from static to dynamic requires a no-glitch methodology, while going from dynamic to static requires that you avoid sampling the bus during precharge.

- Split Phase Logic.

  Rather than wasting the time during phi1, while the dynamic logic is precharging, we managed to get the allocate logic to do some useful work detecting and decoding routing bytes. This has the advantage of stablizing the allocate crosspoint control signals so that there is no glitching during the phi2 evaluate phase.

- Reduced Capacitance Tristate Crosspoint Drivers.

  The crossbar data buses are what is known as *self-loading buses*. This means that sizing up bus drivers increases capacitance in exact proportion to the drive added. After you have sized up to defeat the interconnect capacitance, you are just adding more channel capacitance. Just beefing up the drivers is thus a losing battle. The only way to fight this is to design the tristate drivers with as little channel capacitance loading the bus as possible.

Our original design had a single bidirectional bus for each row and column. This proved to be a bad design decision, because there would be at any given time eight unused drivers or receivers on each bus. Thus, we split the row and column buses each into two unidirectional buses. Then, the tristate drivers were reimplemented using a static gate and a single NMOS or PMOS to the rails. Figure 6-1 shows our final bus driver circuit.

6.0.1 Manchester Style Grant Propagate

The precharge bus scheme used in the crossbar allocate circuits uses dynamic logic for high speed. The full column allocate logic was simulated using SPICE. The feedback inverter thresholds were adjusted for maximum speed. The phi1 precharge transistors were sized down as low

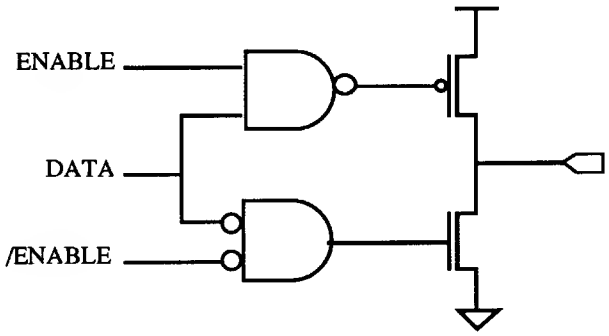


Figure 6-1: These crosspoint drivers present minimal capacitive load to the self-loaded data bus.

as possible to reduce capacitive loading, without prolonging the phi1 precharge phase any longer than needed by the phi1 forward port logic. It is important not to forget that precharging takes time too!

6.0.2 USE Lines

Another critical path was the use status signals which run up the crospoint array columns. These tell each back port whether it has been allocated during a cycle. Speed is of the essence, because the indication that a back port has been grabbed must come as early as possible during a cycle, to allow for the back port logic to enable the output pad drivers. The use line logic serves to create a fast 8 input OR gate of all the individual use signals in each crosspoint in a column. For this OR function, the same precharge-bus with positive feedback circuit is used as that in the allocate grant chain. A latch at the back port serves to isolate the bus, during precharg, from the combinational logic which is watching it in the back port.

6.0.3 Setup on Phi1

The one-shot nature of the dynamic logic in the allocate chain requires that the control signals to the conditional discharge circuits be stable and glitch free when phi2, the evaluate phase, comes around. The best way to insure this was to have those control signals be computed during phi1. This was possible because the allocate decision is made based on information coming in on the pads during phi1. The allocate-early circuits in the forward port tap out the

input data straight from the pads and decode the routing byte destination during phi1; the allocate control signals are stable by phi2.

6.0.4 RESET Logic

One thing to be careful about when designing a complex system is managing the initialization in a complete and deterministic way. The RN1 chip can be fully reset by asserting the INIT signal low and clocking for two or more clock cycles. Since our simulations depended on the entire chip being in a known internal state, we took care at all stages of the design to have a fully working reset sequence for all modules with state.

6.0.5 Clock Distribution

The clock lines were routed as wide metal bus lines, following the the power and ground distribution tree. In addition to the power line distribution tree, the clock is also run directly to the center of the crosspoint array, where it propagates outward from the center.

6.1 Architectural Verification and Testing

With around 50,000 transistors in this design, it was important that we have tools to automate the design and verification process.The design of this chip was aided by several methodologies.

Initial crosspoint logic design was done by Prof. Knight, and served as a starting point around which to build the RN1 chip. The first datasheet was also written by Prof. Knight, which served as a model for the desired behavior and interface of the finished part.

6.2 Architectural Level Simulator

Before further logic design work was started, a simulation at the architectural level was written. Using the Common Lisp Object System, a class was defined for the Transit Network. The network class built its datastructure using instances of a class defined for the RN1 chip. The RN1 chip, in turn, was built with classes modeling the main architectural modules; forward and back ports, crosspoints, and line controllers. The modules responded to phi1 and phi2 clock messages by sending and receiving data through defined register ports.

This object oriented model served to clarify some protocol issues, and led to an initial set of test vector sequences.

6.2.1 Logic Design and Simulation

The logic design was then undertaken. Credit goes to André DeHon for the creation of the first forward and back port state transition diagrams, and in fact for the original idea of splitting the control between independent forward and back port state machines. This division of the logic served to reduce the number of control signals which need to travel across the chip in a single cycle. The state machine diagrams were translated into the input format for the Berkeley BDSyn logic synthesis system. The output of the BDSyn was fed to the MIS II logic optimizer, and the optimized output was then fed to the NS logic-to-schematic library mapper. This led to the automatic creation of schematics using our standard cell logic library, which could be simulated and verified using RSIM and SPICE. The text of the state machine files is given in Appendix B.

The datapaths were designed by hand using the standard cell library, with some additional standard cells to accelerate certain functions, such as the TURN detection logic.The datapaths for the forward and back ports combined with the state machines. The whole module was then placed and routed using the TimberWolf simulated annealing package, and the NS router package. The resulting layout was automatically extracted and graph-matching verified against the original schematic. The purpose of pushing the modules through the layout system early was to get accurate feedback on size and capacitance constraints.

The crosspoint logic was then reworked, with no commitment yet to layout. When we had full chip schematics, we proceeded to write a small test system in Lisp using RSIM primitives. We designed a moderately extensive test suite, which could be run at any time using the single command ":test rchip". The complete test vectors are given in Appendix D.

When we got functional verification, we then turned to optimizing the performance of the chip. Using the NS timing analyzer, PEARL, we found that several critical paths were unacceptably slow. We optimized what we could by resizing drivers, but ultimately had to accept the conclusion that dynamic logic would be necessary to get the speed we wanted.

The redesign of the crosspoints and control logic for dynamic logic was aided by extensive

use of SPICE, which we brought up on the MIT CRAY-II. Figure 6-2 shows an example of a spice run analysis to speed up the carry chain.

When we were reasonably happy with the projected speed of the chip, we started laying out the custom crosspoints. This was aided by running the Berkeley TimberWolf standard cell package on the control logic for the module.

Timing analysis with the new capacitance estimates showed that the crossbar buses still had too much load. This led to splitting them into pairs of undirectional buses, with redesigned tristate drivers.

Power and ground distribution was provided by the composition system, which also handled the intermodule and clock routing. The floorplanner allowed the chip designer to place the major modules in their rough locations as shown. The floorplanner then placed and routed the modules according to the top-level schematic, adding power, ground, and clock buses in a tree distribution. Power and ground were distributed using continuous second layer metal all the way in from bonding pad ring.

The pad ring was specified textually as four NS mask generators, one for each side of the chip. The clock generator, borrowed from Symbolics Ivory Rev3, lives in the bottom pads module.

Final mask verification was unfortunately not complete. Due to problems with the full chip mask extract, the masks were sent out without a full network compare of the CIF file to the schematic network. Partial verification was done to at least make sure there were no power-ground shorts and that the clock lines were not shorted to one another or to the power rails.

6.3 Testing

We constructed a chip test system to interface to the VME bus, which we connected to our XL1200 Lisp machine. The tester was capable of applying stimulation to or reading values from all pins of the chip simultaneously, and could generate a single or two phase clock.

The fabricated parts were packaged in our custom DSPGA372 package, and returned to us. There was a power/gound short in the pad ring, which we removed by scraping with a microprobe, and later with the help of the submicrometer technology group at Lincoln Labs.

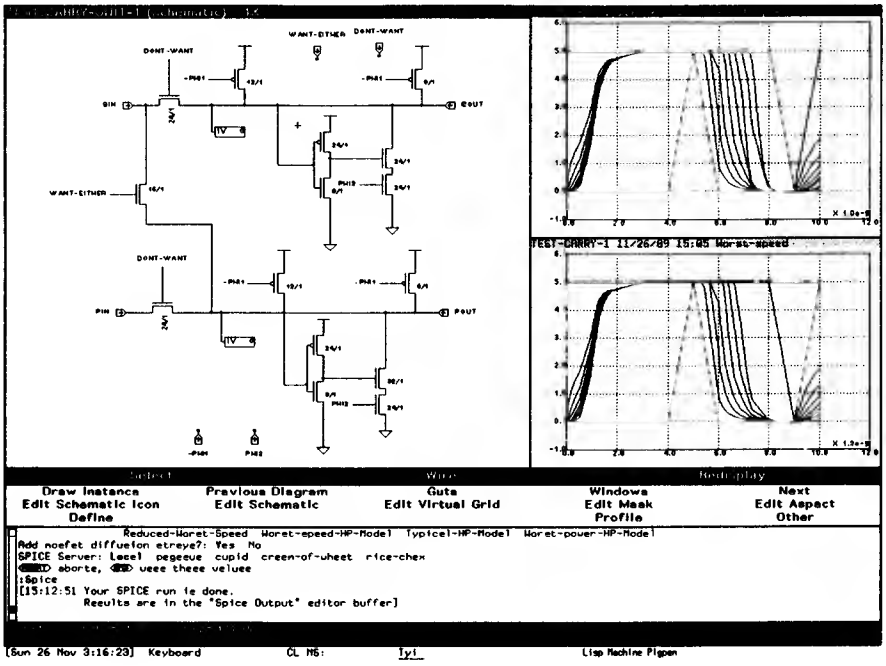


Figure 6-2: Spice Simulation of the dynamic Allocate Logic

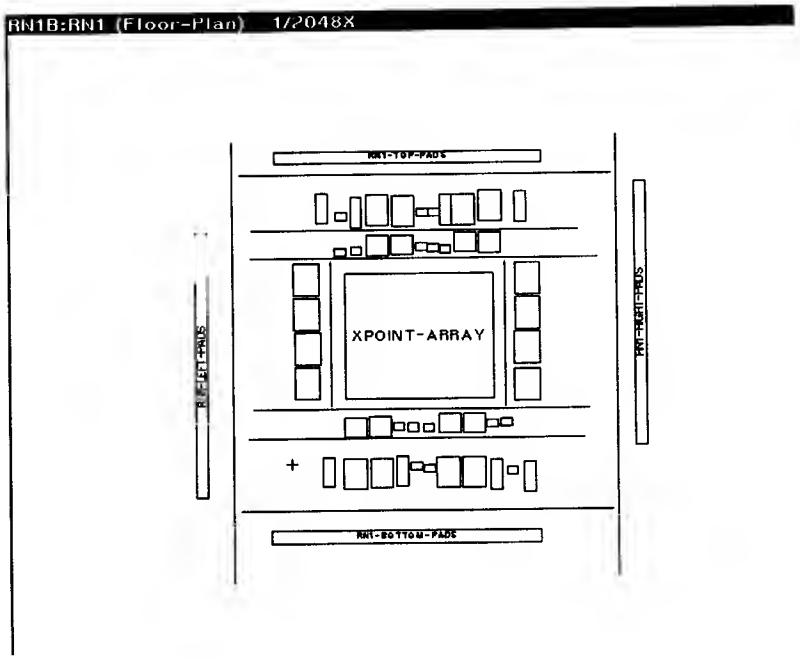


Figure 6-3: RN1 Chip Floorplan

Thanks go to Scott Doran at Lincoln for helping to rework half a dozen chips.

Chapter 7

Chip Performance, Bugs, and Future Improvements

We received twenty packaged parts from MOSIS, and a handful of unpackaged die. To test the chips, we built a tester board to attach to the VME bus of our XL1200 Lisp Machine. This allowed us to apply data, clocks, and control signals to the RN1 chips, and to capture the chip responses.

Our custom button-board packaging required special mechanical work to build a test board fixture and cabling. The button board connectors themselves have been quite reliable, although they are very fragile when not installed in the system, and require careful handling.

7.1 Functional Tests

The chips were partially functional; they were able to self-route connections, and to recognize the commands to drop or reverse a connection. The presence of a mask error in the pad ring prevented the control bit of BKPORT7 from going low, thus preventing that port from being useful in a network configuration.

While it was possible to make simple connections through the chip, a race condition was discovered in the `line-ctrl` logic, which allowed the line allocation data from a previous cycle

Chapter 7

Chip Performance, Bugs, and Future Improvements

We received twenty packaged parts from MOSIS, and a handful of unpackaged die. To test the chips, we built a tester board to attach to the VME bus of our XL1200 Lisp Machine. This allowed us to apply data, clocks, and control signals to the RN1 chips, and to capture the chip responses.

Our custom button-board packaging required special mechanical work to build a test board fixture and cabling. The button board connectors themselves have been quite reliable, although they are very fragile when not installed in the system, and require careful handling.

7.1 Functional Tests

The chips were partially functional; they were able to self-route connections, and to recognize the commands to drop or reverse a connection. The presence of a mask error in the pad ring prevented the control bit of BKPORT7 from going low, thus preventing that port from being useful in a network configuration.

While it was possible to make simple connections through the chip, a race condition was discovered in the `line-ctrl` logic, which allowed the line allocation data from a previous cycle to appear momentarily at the start of a clock cycle. This would not be a problem for a chip with all static logic, but dynamic precharge logic is not forgiving of glitches. The result is

that the RN1 chip cannot route a new connection reliably on a cycle which occurs immediately after a event which has changed the state of the line-control status lines `p0, p1, s0, s1`. Unfortunately, the random bit generator from the forward ports can change the state of these lines on any cycle. Thus, the chip is not usable in its present form for building a network. These bugs are understood, and will be fixed in the next revision, along with several improvements which we have developed since RN1 was designed.

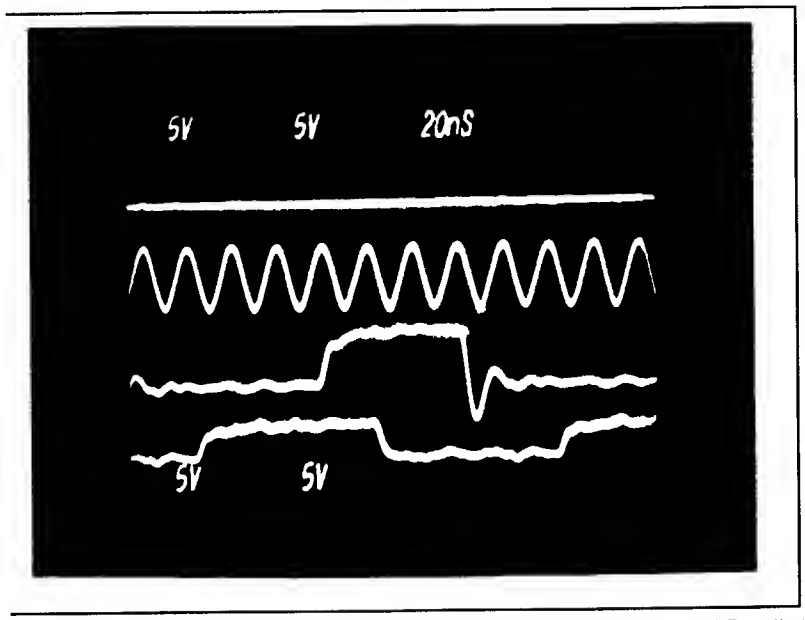
7.2 Performance

We are still in the process of testing the chips at speed, to find the maximum clock frequency at which they can operate. This is complicated by the lack of a real high speed chip tester. We have made several test boards using high speed PAL devices to stimulate the chip at high clock speeds.

One of the most significant limitations to the clock speed which we can cycle the part seems to be a long phase delay in the internal two-phase clock generator. The delay through the allocate cycle, as measured from the falling edge of the input clock, seems to be about 33ns. This delay is actually pessimistic, when the clock generator phase delay is taken into account. Figure 7-1 shows a scope trace of the RN1 part opening a connection and transmitting data at a clock frequency of 55Mhz. If the on chip clock skew could be reduced, this would indicate the the part could be clocked reliably at 50Mhz. Thus far, lacking a high speed tester, we have run very simple open-close connection tests through the part at a sustained 50Mhz clock rate. The HP CMOS 5v pads account for about 10ns of this delay, so that a move to faster I/O pads could speed the cycle time up considerably.

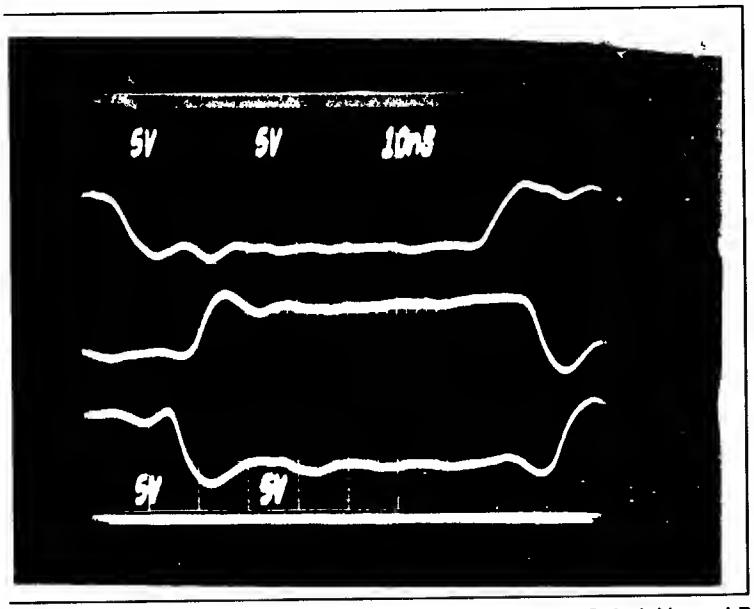
Figure 7-2 shows the phase relationship between the external clock and the internal two-phase clocks which are derived from it. The delay between external clock falling and internal phi2 rising seems to be about 13ns. This clearly needs to be optimized in the next revision.

Figure 7-3 shows the delay from the internal pad loopback circuit. This circuit consists of an input pad internally looped back to an output pad. The total on-chip to off-chip delay measures around 10ns.



Threshold 6, Contrast 1, Brightness 9, Halftone Pattern Spiral, Normal Detail 1/
55 Mhz Allocate-Drop

Figure 7-1: RN1 clocking a connection at 55 Mhz



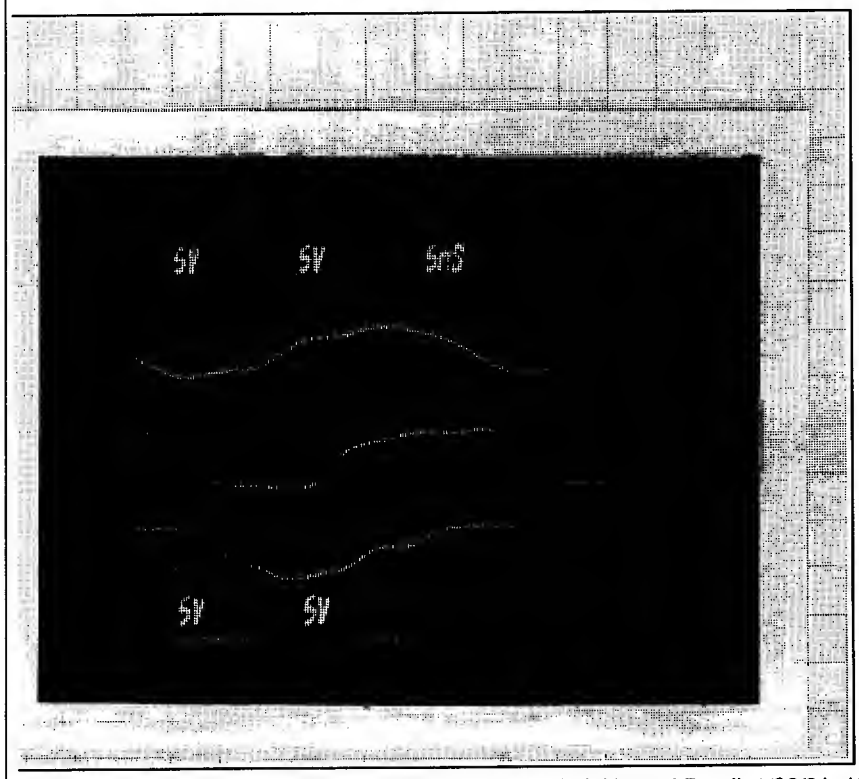
ext clk

φ2

φ1

Threshold 12, Contrast 1, Brightness 9, Halftone Pattern Spiral, Normal Detail 1/
Clock Generator

Figure 7-2: Relation between external clock and internal clocks.



shold 9, Contrast 3, Brightness 9, Halftone Pattern Spiral, Normal Detail 1/22/91 12:17
Loopback Test

Figure 7-3: Pad internal loopback test

7.3 Conclusions

The RN1 chip as conceived and executed has achieved its stated design purpose in many areas. Although the speed goal of 100MHz performance in CMOS technology now seems to be a little too agressive, the RN1 chip prototype has allowed us to verify many aspects of our system design. The integrated three-dimensional packaging technology shows great promise, and we are now designing the boards for a 64 port Transit network using this technology. The RN1 has provided proof of concept of our crossbar design and routing protocol, and will, with the appropriate bug fixes and improvements, allow us to build our interprocessor communication switch. The next VLSI chip will be considerably easier to produce, having gone through the excercise once already.

7.3.1 Simulation: Models vs. Reality

One of the lessons to be learned from this design experience is that it is very important to be aware of the limits of the circuit simulators and models which are used in our design tools. The dynamic logic race condition was not apparent in our simulations using the RSIM switch-level simulator. A full simulation in SPICE would have in fact caught the problem, but that was deemed impractical, with the tools we were running. I simulated the dyanmic logic paths in SPICE to verify that they were correct, but failed to do complete simulation of the interface circuits between the static and dynamic logic. The important point is to understand where trouble is likely to occur, and concentrate design and verification effort in those places.

7.3.2 Test Vectors

We were bitten by a bug which could easily have been caught by our verification methodology had we done a better job of creating test vectors for the simulation. The lesson here is that it is very important to create a complete and exhaustive set of tests to verify that the system performs the desired function. The bug I am referring to is the allocate-early reset bug. This was a bug which was introduced late in the design cycle, when I decided to speed up the system by adding an independent state machine in the forward port to help do routing allocation during the first phase of the two phase clock cycle. The bug went unnoticed because our test suite

did not happen to have a long enough sequence of sample traffic to get the forward port into a state which would cause an error.

Our test suite was deficient because we convinced ourselves that we had designed an exhaustive set of tests, by reasoning out all the possible states of the chip and testing each of them in one way. The problem with this methodology is that when the implementation changed late in the design cycle, our early reasoning as to the operation of the chip was no longer complete. The correct thing to have done would have been to get fairly large and extensive traffic sequences from actual simulations, and use them in addition to our carefully constructed test cases. This would serve to create a background which could fill in the blind-spots left by our design tunnel-vision.

7.4 Future Improvements

We are currently working on the next revision of the RN1 part, RN1B. Aside from known bug fixes, it will have the following architectural improvements

- *CRC-CCITT Checksum Unit*

  The new CRC generator (Appendix A) uses the CRC-CCITT polynomial checksum generator.

- *External Random Inputs*

  In order to make the part more testable, and pave the way for wider datapath networks, we have made the source of random bits for the allocate preference in the line-control units source from an external 4 bit bus. The RN1B part also generates a single `RANDOM` output, which can be fed to neighboring parts.

- *TURN Byte code now depends only on top two bits of data*

  In order to allow extra control codes, we have made the `TURN` code depend only on the following logic equation: -CBIT*DATA< 7 >*DATA< 6 >. This allows three other valid code words at the end of a message data sequence besides the TURN command.

[DeHon 90e] details a more complete list of suggested improvements for future chip revisions.

7.5 Possible Architectural Extensions

- The idea for a *quick-drop* signal has been proposed. This would consist of one extra control signal per forward port, which would immediately signal the previous chip if a newly allocated connection was blocked. This would alleviate the need to wait until a **TURN** command has been issued to discover that a connection is blocked. The quick teardown of the blocked partial path through the network would relieve congestion and allow the sender to retry sooner.

- The quick-drop pin is only used once at the start of each allocate cycle. During the rest of the time, it could be used for other routing supervisory purposes; congestion information could be propagted back through this control signal to previous chips in the network, to assist the line-control units to choose a benficial primary output channel.

- *Double clock speed message tranmission*

  The allocate cycle is the critical path in chip timing; once a data connection has been opened, it takes a fraction of the allocate time to simply pass the data through the forward port, crosspoint, and back-port drivers. It should be possible to break it into two piplined phases, and increase the clock speed and thus the speed of data transmission.

Bibliography

[Barber 88] Frank E. Barber, et. al, *A 64x17 Non-Blocking Crosspoint Switch*, IEEE International Solid-State Circuits Conference, pg. 115-116, 1988.

[Benes 65] V. E. Beneš, *Mathematical Theory Of Connecting Networks And Telephone Traffic* Academic Press, New York.

[Campbell 87] Joe Campbell, *C Programmer's Guide to Serial Communications* Howard W. Sams & Company, 1987

[Cherry 86] J. Cherry, *PEARL CMOS Timing Analyzer* 8th Design Automation Conference, California.

[Chong 90] Fred Chong, *Analog Techniques for Adaptive Routing on Interconnection Networks*, MIT Transit Note #14

[Comer 88] Douglas E. Comer, *Internetworking With TCP/IP*, Prentice Hall, 1988

[Dally 87] W. J. Dally, C. L. Setiz *Deadlock-Free Message Routing in Multiprocessor Interconnection Networks*, IEEE *Transactions on computers*, Vol. C-36, No. 5, May 1987.

[DeHon 90a] André DeHon, Tom Knight, Henry Minsky, *Fault Tolerant Design for Multistage Routing Networks*, MIT AI Lab Memo 1225

[DeHon 90b] André DeHon, *Modular Bootstrapping Transit Architecture*, MIT Transit Note #17

[DeHon 90c] André DeHon, *Network Level Transactions*, MIT Transit Note #19

[DeHon 90e] André DeHon, *RN1: Things To Improve Upon*, MIT Transit Note #1

[Egozy 90e] Eran Egozy, *Transit Network Fault Tolerance Simulations*, MIT Transit Note #39

[Gigabit 1988] Gigabit Logic, *10G051 16x16 Crosspoint Switch*, datasheet, pg. 1-106:110

[Gottlieb 86] A. Gottlieb, *An Overview of the NYU Ultrcomputer Project.* Ultracomputer Note #100, Ultracomputer Research Laboratory, New York University, Division of Computer Sciences

[Gottlieb 89] A. Gottlieb, G. Almasi, *Highly Parallel Computing*, Benjamin/Cummings Publishing Company, 1989

[Hui 90] Joseph Y. Hui, *Switching And Traffic Theory For Integrated Broadband Networks*, Kluwer Academic Publishers, 1990.

[Knight 89a] Thomas Knight, Jr. *Technologies for low latency interconnection switches*, In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pp. 351-358, June 1989

[Knight 89b] Thomas Knight, Jr., Alexander Krymm*A Self-Terminating Low Voltage Swing CMOS Output Driver, IEEE Journal of solid-state circuits*, 23(2), April 1988

[Kruskal 86] C. P. Kruskal and M. Snir, *The Performance of Multistage Interconnection Networks for Multiprocessors.* IEEE Trans. Comput., vol C-32, pp 1091-1098, 1983.

[Leighton 89] Tom Leighton, Bruce Maggs, *Expanders Might Be Practical: Fasr Algorithms for Routing Around Faults on Multibutterflies*, 30th Annual Symposium on Foundations of Computer Science, pp. 384-389, 1989

[Patel 81] Janak H. Patel, *Performance of Processor-Memory Interconnections for Multiprocessors*, IEEE Transactions on Computers, pp 771-780 Vol. C-30, No. 10, October 1981.

[Siegel 90] Howard. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing*, ISBN 0-669-03594-7 McGraw-Hill, 1990

[Tanenbaum 81] Andrew S. Tanenbaum, *Computer Networks*, Prentice-Hall, 1981

[Tanenbaum 87] Andrew S. Tanenbaum, *Operating Systems Design and Implementation*, ISBN 0-13-637406-9 Prentice-Hall, 1987.

[Transit 90] MIT Transit Group, *DSPGA372*, *BB372*, Transit Group Packaging Documentation, 1990

[Berkeley 64] UCB Electrical Engineering Department, *SPICE* University Of California, Berkeley.

Appendix A

Checksum Generator

The checksum function used was based on a modem pseudorandom bit scrambler circuit. This turned out to be very awkward to implement, and did not have especially good performance as a check for bit errors. Also, the circuit as designed had the annoying property of clocking once in between sending the first and second byte of the CRC during a turn. This made emulating the behavior of the checksumin hardware at the network interface needlessly complex.

The checksum is computed using a 17 bit feedback shift-register into which the 8 bit data is XORed. The feedback taps are at bits 4 and 16.

The checksum is emitted from the RN1 in two successive bytes: 6 bits in the STATUS byte, and the remaining 8 bits in the CHECKSUM byte. During this sequence, the checksum unit is clocked once, so the second half of the checksum will be the value of the bottom 8 bits of the shift register, after one cycle, with the previous STATUS word having been presented as the data input during the last cycle. The following two lisp functions, rchip-status and rchip-checksum, reflect this one cycle latency. rchip-checksum computes its value using the sequence of data words plus the status word which would have been computed one cycle before it. An example from the RN1 test vectores illustrates the actual checksum values.

The following lisp code implements the checksum circuit in RN1, rev 1.

```
;;  Compute-checksum computes the checksum
;;
;; inputs: data-bytes :: list of 8-bit data bytes
;;
;; outputs: (status checksum) :: two 8 bit bytes
;;
;; The Checksum register is a 17 bit shift reg with feedback
;; from bits 4 and 16. It is best described by this piece of code.
(defun compute-checksum (data-bytes)
```

```
  (declare (values status checksum))
  (let ((checksum 0))
    (loop for data in data-bytes
  for feedback-bit = (logxor (ldb (byte 1 4) checksum)
 (ldb (byte 1 16) checksum))
  for feedback-byte = (logand #o377
      (logxor data (dpb (ldb (byte 7 0) checksum)
(byte 7 1)
feedback-bit)))
  do (setq checksum
   (dpb feedback-byte (byte 8 0)
(logand #o377777
(lsh checksum 1))))
  finally (return (values (ldb (byte 8 0) checksum)
  (ldb (byte 8 8) checksum))))))
;; This has to OR in the top 6 bits of status with use1,use0
(defun rchip-status (data-bytes use1 use0)
  (multiple-value-bind (status ignore)
      (compute-checksum data-bytes)
    (values (logior (lsh (ldb (byte 6 2) status) 2)
    (lsh use1 1)
    use0
    #x100 ;add in the control bit
    ))))
;; The checksum comes one clock cycle later, so we have to act as if we clocked
;; the checksum unit once more, with input data equal to the status byte which
;; was driven out last cycle!
(defun rchip-checksum (data-bytes use1 use0)
  (multiple-value-bind (ignore checksum)
      (compute-checksum (append data-bytes
  (list (rchip-status data-bytes use1 use0))))
    (values (logior #x100 ;control bit
    checksum))))
```

This example test vector shows what the expected checksum and status bytes are for a connection which was opened with routing byte $1C0, and had data $155 $122 $133 $1FF sent through it.

Note that checksum comes exactly one cycle after status.

```
(defun test-turn1 ()
  (remark "~&Opening connection from port 0 to #x1C0")
  (test-core-step :fwds '(#x1C0 0 0 0 0 0 0 0)
  :expected-bckout '(0 0 0 0 0 0 0 #x1c0))
  (remark "~&Sending data ")
  (test-core-step :fwds'(#x155 0 0 0 0 0 0 0)
  :expected-bckout '(0 0 0 0 0 0 0 #x155))
  (test-core-step :fwds '(#x122 0 0 0 0 0 0 0)
  :expected-bckout '(0 0 0 0 0 0 0 #x122))
  (test-core-step :fwds '(#x133 0 0 0 0 0 0 0)
  :expected-bckout '(0 0 0 0 0 0 0 #x133))
  (test-core-step :fwds '(#x1FF 0 0 0 0 0 0 0)
:expected-bckout '(0 0 0 0 0 0 0 #x1FF))
  (remark "~&Sending turn byte, expecting status ")
  (let ((status (rchip-status '(#x155 #x122 #x133 #x1FF) 1 0))
(checksum (rchip-checksum '(#x155 #x122 #x133 #x1FF) 1 0)))
  (test-core-step :fwds '(#x0FF 0 0 0 0 0 0 0)
  :expected-fwdens '(1 0 0 0 0 0 0 0)   ;fd port becomes an output
```

```
  :expected-fwdout (list status 0 0 0 0 0 0 0) ;status
  :expected-bckens '(1 1 1 1 1 1 1 1)
  :expected-bckout '(0 0 0 0 0 0 0 #x0FF)) ;chip is driving data fd and bk.
  (remark "~&chksum")
  (test-core-step :expected-fwdens '(1 0 0 0 0 0 0 0)
  :expected-fwdout (list checksum 0 0 0 0 0 0 0 ) ;checksum
  :expected-bckens '(1 1 1 1 1 1 1 0))) ;back port is an input now
  ;;
;  (break)
  (remark "~&Driving data #x123 into back port")
  (test-core-step :expected-bckens '(1 1 1 1 1 1 1 0)
  :bcks '(0 0 0 0 0 0 0 #x123)
  :expected-fwdens '(1 0 0 0 0 0 0 0)
  :expected-fwdout '(#x123 0 0 0 0 0 0 0))
```

Appendix B

State Machine Code

B.1 Forward Port State Machine

The following state-machine programs are written to be input to the Berkeley BDSYN finite-state machine compiler.

```
MODEL  in  outEnable, toXbar, stat, chk, net, allocate, rst,
 idlepat, NxtState<2:0> =
PrvState<2:0>, sw, cb, turn, use0, use1,init ;

! RST/RESET
! INPUT.L = -INPUT
! state mnemonics
CONSTANT idle = 0, swallow = 1, fwd = 2, noturn=3,
status = 4, close = 5, backward = 6, bck2 = 7;

ROUTINE input_fsm;

net = 1;    ! if not otherwise specified, mux the pad output from net
toXbar = 1;    ! by default, data from flops is pushed into xbar
outEnable = 0;  ! by default, pad is an input pad.

IF init THEN
   BEGIN
     NxtState = idle;
     rst = 1;
   END

ELSE

SELECT PrvState FROM

[status]:
BEGIN

toXbar = 0;  ! get data from Xbar
outEnable = 1;  ! enable pad as output
        net=0;chk=1;

        IF (use0 NOR use1) THEN
            NxtState = close
        ELSE
            NxtState = backward;

        END;

[close]:
```

```
BEGIN
toXbar=0;
outEnable = 1;
        idlepat = 1;  ! send a drop to the previous chip
        NxtState = idle;
        END;

[backward,bck2]:
BEGIN
toXbar = 0;  ! get data from Xbar
outEnable = 1;  ! output enabled
        IF turn THEN
           NxtState = noturn
        ELSE
           IF cb THEN
              NxtState = backward
           ELSE
                BEGIN
                   NxtState = idle;
                   RST = 1; net=0;
                   idlepat = 1;
       END;
        END;

[idle]:
BEGIN
        IF (NOT cb) THEN
           NxtState = idle
        ELSE
           IF sw THEN
              NxtState = swallow
           ELSE
                BEGIN
                   NxtState = fwd;
                   allocate = 1;
              END;
        END;

[swallow]:
BEGIN
        allocate = 1;
        NxtState = fwd;

        END;

[fwd]:
BEGIN
        IF turn THEN
            BEGIN
       outEnable = 1;  ! start driving status out
              net=0; stat = 1;
              NxtState = status;
            END
        ELSE
           IF cb THEN
              NxtState = fwd
           ELSE
                BEGIN
                  NxtState = idle;
rst = 1;
              END;
        END;

[noturn]:
BEGIN
```

```
outEnable = 0;  ! become an input
NxtState = fwd;
        END;

   ENDSELECT;

ENDROUTINE;
ENDMODEL;
```

B.2 Back Port State Machine

```
MODEL  out   outEnable, idlepat, cbit, drop,
NxtState<2:0> =
PrvState<2:0>, use, fwdturn, bckturn, fwddrop, bckdrop;

! state mnemonics
CONSTANT idle = 0, idle2 = 1, goidle = 2, goidle2 = 3, fwd = 4,
         turnfwd = 5, backward = 6, turnbck = 7;

ROUTINE output_fsm;

outEnable = 1; ! default to being an output
idlepat = 1;

SELECT PrvState FROM

[idle,idle2,goidle,goidle2]:
BEGIN
        idlepat = 1;
        IF (use) THEN
            BEGIN
idlepat  =  0;
                NxtState  =  fwd;
            END
        ELSE
            NxtState  =  idle;
        END;

[fwd]:
BEGIN
idlepat = 0;
IF fwddrop THEN
    BEGIN
drop  =  1;
NxtState =  idle;
    END
    ELSE
            IF fwdturn THEN
                NxtState  =  turnbck
            ELSE
                NxtState  =  fwd;
        END;

[backward]:
BEGIN
outEnable = 0;  ! be an input pad
```

```
IF bckdrop THEN
  BEGIN
     drop  =  1;
              idlepat  =  1;
     outEnable  =  1;   ! become an output, driving idle
     NxtState  =  idle;
   END
ELSE
        IF bckturn THEN
            BEGIN
 outEnable  =  1;
                 cbit  =  1; !hold back connection open during turn
                 NxtState  =  turnfwd;
            END
        ELSE
   NxtState  =  backward; !connection still open.
        END;

[turnfwd]:
BEGIN
        cbit = 1; ! keep external connection open
        NxtState  =  fwd;
        END;

[turnbck]:
BEGIN
outEnable  =  0;  ! become an input pad
        NxtState  =  backward;
        END;

   ENDSELECT;

ENDROUTINE;
ENDMODEL;
```

Appendix C

RN1 Revision 2 CRC

The second revision of the RN1 chip, RN1b, will use the CRC-CCITT 16 bit polynomial cyclic redundancy check, as implemented by the `bytestep()` function show below. [Tanenbaum 81] states the properties of the CRC-CCITT 16 bit polynomial checksum. These include catching

- All single-bit errors
- All double-bit errors
- All errors with odd number of bits
- All burst errors shorter than 16 bits
- 99.9969 percent of 17 bit burst errors
- 99.9984 percent of all other burst errors

The specifications for the RN1b chip explicitly state that the checkword generator is not to be run in between sending the first and second bytes (`status, checksum` of the CRC during a turn.

```
/*****************************
 *
 * CRC-CCITT generator simulator for byte wide data. Derived from from Joe
 * Campbell's
 * ' C Programmer's Guide to Serial Communications '
 *
 * (c) Henry Minsky 1990
 *
 * This simulates the XOR circuit used to generate the modulo-2
 * remainder by dividing the message data by the CRC-CCITT generator
 * polynomial.
 *
 * CRC-CCITT = x^16 + x^12 + x^5 + 1
 *
 * The xor tree was generated by me by hand.
 *
 *****************************/

#include <stdio.h>
typedef unsigned short BIT16;
int message[] = { 0xC3, 0x66, 0xF9, 0x55, 0x12, 0x33, 0x09, 0x4f, 0x23, 0x77};
```

```
BIT16 p_div(BIT16 data, BIT16 divisor, BIT16 remainder);
BIT16 crchware(BIT16 data, BIT16 genpoly, BIT16 accum);

main()
{
int a[16];
int d[8];
int i, bit, j;
BIT16 acc = 0;

printf("sizeof(BIT16) = %d\n", sizeof(acc));
for (i = 0; i < 10; i++) {
acc = crchware((BIT16) message[i], (BIT16) 0x1021, acc);
printf("crchware acc = 0x%04x\n", acc);
}

/* clear the accumulator shiftregister */
for (i = 0; i < 16; i++)
a[i] = 0;

for (i = 0; i < 10; i++) {
/* fill data array with data */
for (bit = 0; bit < 8; bit++)
if ( message[i] & (1 << bit) )
d[bit] = 1;
else
d[bit] = 0;

bytestep(a, d);

acc = 0;
for (j = 0; j < 16; j++)
acc += a[j] << j;
printf("CRC value = 0x%04x\n", acc);

}

}

bytestep(BIT16 a[], BIT16 d[])
{ BIT16 t[16];

t[0] = a[8] ^ a[12] ^ d[4] ^ d[0];

t[1] = a[9] ^ a[13] ^ d[5] ^ d[1];

t[2] = a[10] ^ a[14] ^ d[6] ^ d[2];

t[3] = a[11] ^ a[15] ^ d[7] ^ d[3];

t[4] = a[12] ^ d[4];

t[5] = a[13] ^ a[8] ^ a[12] ^ d[5] ^ d[4] ^ d[0];

t[6] = a[14] ^ a[9] ^ a[13] ^ d[6] ^ d[5] ^ d[1];

t[7] = a[10] ^ a[14] ^ a[15] ^ d[2] ^ d[6] ^ d[7];

t[8] = a[0] ^ a[11] ^ a[15] ^ d[7] ^ d[3];

t[9] = a[1] ^ a[12] ^ d[4];

t[10] = a[2] ^ a[13] ^ d[5];

t[11] = a[3] ^ a[14] ^ d[6];

t[12] = a[4] ^ a[15] ^ a[8] ^ a[12] ^ d[7] ^ d[4] ^ d[0];

t[13] = a[5] ^ a[9] ^ a[13] ^ d[5] ^ d[1];
```

```
t[14] = a[6] ^ a[10] ^ a[14] ^ d[6] ^ d[2];
t[15] = a[7] ^ a[11] ^ a[15] ^ d[7] ^ d[3];
{
int i;
for (i = 0; i < 16; i++) a[i] = t[i];
}
}

/* models polynomial division */
BIT16 p_div(BIT16 data, BIT16 divisor, BIT16 remainder)
{
static BIT16 quotient, i;
for (i = 8; i > 0; i--) {
quotient = remainder & 0x8000;
remainder <<= 1;
if ( (data <<= 1) & 0x100)
remainder |= 1;
if (quotient)
remainder ^= divisor;
}
return (remainder);
}
/* models crc hardware (minor variation on polynomial division algorithm) */
BIT16 crchware(BIT16 data, BIT16 genpoly, BIT16 accum)
{
static BIT16 i;
data <<= 8;
for (i = 8; i > 0; i--) {
if ((data ^ accum) & 0x8000)
accum = (accum << 1) ^ genpoly;
else
accum <<= 1;
data <<= 1;
}
return (accum);
}
```

Appendix D

Test Vectors

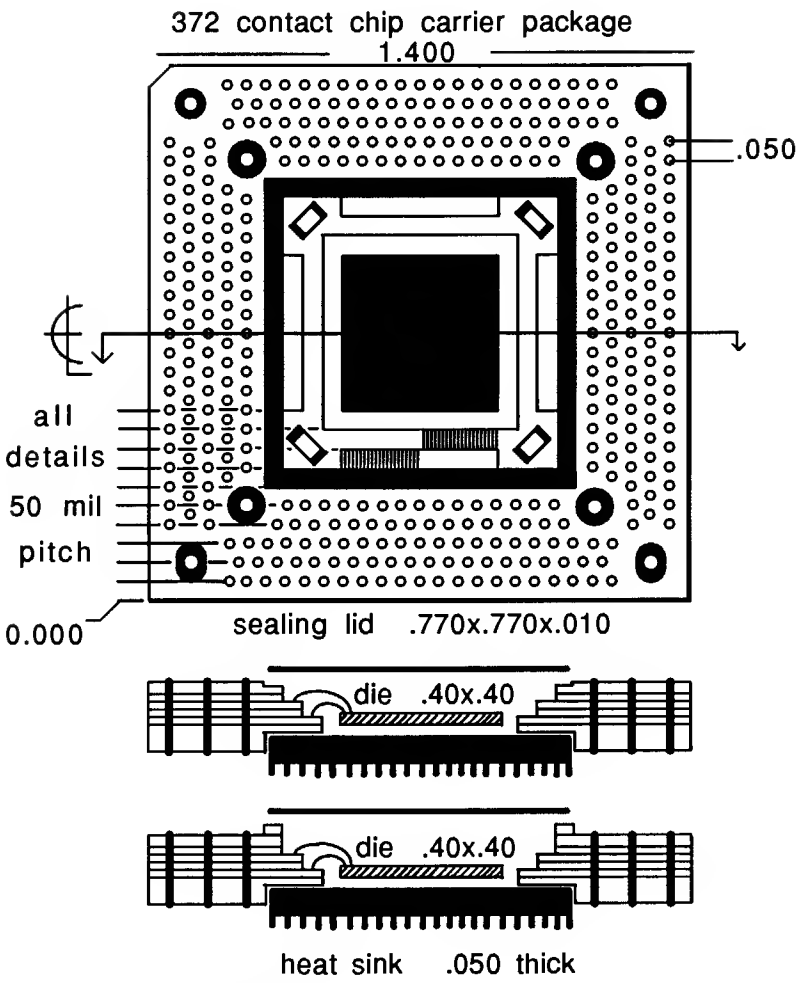
The test vector sets used in RN1, such as they are, are available on Internet host REAGAN.AI.MIT.EDU:>transmit>sim>rchip-test-vectors.

Appendix E

Datasheet

(The RN1 Datasheet is included here)

CHIP CARRIER REV 1



layer 1 top pads
layer 2 ground plane
layer 3 level 2 bonding
layer 4 +5volt power plane
layer 5 level 1 bonding
layer 6 +1volt power plane
layer 7 bottom pads

o .030 pad .....012 drill.....plated through hole (can be plated shut) 372 places

.078 diameter hole -0.000 +.002 non-plated 2 places

.078 slot -0.000 +0.002 .100 long non-plated 2 places

.100 diameter hole +-.002 4 places

type R16 chip capacitor .050x.100x.050 max 4 places

bonding pads 400 locations (shown partially) 2 levels of 50 per side

This blank page was inserted to preserve pagination.

CS-TR Scanning Project
Document Control Form

Date : 6/15/95

Report # AI-TR-1284

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 111(118-IMAGES)
Not to include DOD forms, printer instructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☐ Other:_____

Check each if included with document:

☒ DOD Form ☐ Funding Agent Form ☒ Cover Page
☒ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number): PAGE FOLLOWING TITLE PAGE

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:

(A) IMAGE MAP (1-2) UNNUMBERED TITLE AND BLANK PAGES
(3-110) PAGES NUMBERED 3-110
(111) UNNUMBERED PAGE (DIAGRAM)
(112-118) SCANCONTROL, COVER, SPINE, DOD, TARGETS (3)
(B) CUT&PASTE FIGS ON PAGES 38, 85, 86, 90-92.

Scanning Agent Signoff:

Date Received: 6/15/95 Date Scanned: 6/15/95 Date Returned: 6/22/95

Scanning Agent Signature: Michael W. Cook

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1991	technical report

4. TITLE AND SUBTITLE
A Parallel Crossbar Routing Chip for a Shared Memory Multiprocessor

5. FUNDING NUMBERS
N00014-88-K-0825
N00014-85-K-0124

6. AUTHOR(S)
Henry Minsky

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Artificial Intelligence Laboratory
545 Technology Square
Cambridge, Massachusetts 02139

8. PERFORMING ORGANIZATION REPORT NUMBER
AI-TR 1284

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
Information Systems
Arlington, Virginia 22217

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AD-A 259 489

11. SUPPLEMENTARY NOTES
None

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution of this document is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis describes the design and implementation of an integrated circuit and associated packaging to be used as the building block for the data routing network of a large scale shared memory multiprocessor system.

A general purpose multiprocessor depends on high-bandwidth, low-latency communications between computing elements. This thesis describes the design and construction of RN1, a novel self-routing, enhanced crossbar switch as a CMOS VLSI chip. This chip provides the basic building block for a scalable pipelined routing network with byte-wide data channels. A series of RN1 chips can be cascaded with no additional internal network components to form a multistage fault-tolerant routing switch. The chip is designed to operate at clock frequencies up to 100Mhz using Hewlett-Packard's HP34 1.2μ process. This aggressive performance goal demands that special attention be paid to optimization of the logic architecture and circuit design.

14. SUBJECT TERMS (key words)			15. NUMBER OF PAGES
parallel processing computer architecture multistage routing network			112
			16. PRICE CODE $8.00
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**